UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Dissertação de Mestrado

RelP

Um *middleware* para publicação e consulta de dados relacionais através de visões SQL/XML

CLAYTON MACIEL COSTA

FORTALEZA/CE
2009

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

RelP

Um *middleware* para publicação e consulta de dados relacionais através de visões SQL/XML

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Vânia Maria Ponte Vidal

CLAYTON MACIEL COSTA

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

RelP

Um *middleware* para publicação e consulta de dados relacionais através de visões SQL/XML

Clayton Maciel Costa

Orientadora: Profª. Drª. Vânia Maria Ponte Vidal

Aprovada em __/__/_____

BANCA EXAMINADORA

Prof.ª Dr.ª Vânia Maria Ponte Vidal (Orientadora)
Universidade Federal do Ceará – UFC

Prof. Dr. Fabio André Machado Porto
Laboratório Nacional de Computação Científica – LNCC

Prof. Dr. José Antonio Fernandes de Macêdo
Universidade Federal do Ceará – UFC

AGRADECIMENTOS

De diversas formas, e em momentos diferentes, muitas pessoas direta ou indiretamente, contribuíram para o desenvolvimento deste trabalho.

A Deus, por tudo e principalmente por ter iluminado meu caminho durante essa oportunidade de crescimento e aprendizado, e a Ele entrego minha vida.

Aos meus pais, que sempre foram meu amparo, porto seguro, durante todos estes anos da minha vida e que fizeram de tudo para que eu trilhasse pelo melhor caminho.

A minha querida esposa Ceiça e ao meu filho Pedro Davi, meus amores e grandes amigos, que estiveram do meu lado nos momentos difíceis, com carinho e alegria, e me ensinaram a persistir nos meus sonhos. Obrigado por existirem na minha vida.

As minhas irmãs, Vânia e Lívia, por terem agüentado o meu mau humor nos momentos difíceis e por compartilharem dos momentos de felicidade.

A minha querida orientadora Vânia Vidal pela persistência, crença, incentivo, ensinamentos e dedicação muito bem empregados no acompanhamento e orientação desta dissertação.

Ao meu amigo e co-orientador Fernando Lemos pela crença, paciência, ensinamentos e dedicação no acompanhamento desta dissertação.

Ao Professor Dr. Fabio André Machado Porto e ao Professor Dr. José Antonio Fernandes de Macêdo, por terem aceitado o convite para participar da banca examinadora e por colaborarem para a melhoria deste trabalho.

A minha avó e meu avô, por suas orações e ensinamentos constantes.

Aos meus tios e tias, Lenilson, Lázaro, Neto, Lindalva, Lindoci, Lenilda, Marta, Netinha pela preocupação constante. Aos meus primos, que me incentivam sempre.

A minha amiga Bernadette Farias Lóscio por ter me ouvido, acredito e me incentivado durante o mestrado.

Aos meus queridos amigos Bruno, Fernando Wagner, Danusa, Luiz, Hélio, Lívia, Ticiana, Ticianne, Henrique, Lígia, Ângela, João, Eveline, Artur, Vinícius, Juliana, André, Márcio, Jorge, Jânio, João Paulo e todos que já passaram pela minha vida e me ajudaram com a sua amizade, idéias e também com conselhos.

A Universidade Federal do Ceará pela oportunidade do grande crescimento profissional e intelectual.

Aos professores e funcionários do Mestrado em Ciência da Computação da Universidade Federal do Ceará pelo apoio à realização do curso.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo auxílio financeiro que possibilitou a realização deste trabalho.

RESUMO

XML se estabeleceu como o formato padrão para a publicação, integração e troca de dados na Web. Entretanto, a imensa maioria dos dados de negócio, incluindo dados de aplicações Web, continua sendo armazenada em SGBDs relacionais.

Atualmente, em muitos cenários, a mesma informação é armazenada e consultada em SGBDs relacionais, mas exibida e compartilhada no formato XML. Devido à larga utilização de aplicações Web que utilizam bancos de dados relacionais juntamente com a tecnologia XML, e ainda, devido às diferenças substanciais entre o modelo relacional e o modelo XML, publicar dados relacionais no formato XML é um problema de extrema relevância e não trivial. Além disso, as abordagens existentes não resolveram esse problema de forma satisfatória.

Neste trabalho, tratamos o problema de publicação de dados relacionais no formato XML. Para tanto, apresentamos um *middleware*, chamado ***RelP***, para publicação e consulta de dados relacionais através de visões SQL/XML. No ***RelP***, um usuário pode realizar, via *serviço Web de acesso a dados*, consultas sobre uma visão SQL/XML. A consulta é traduzida em uma consulta SQL/XML equivalente definida sobre o esquema do banco de dados relacional. *Serviços Web de acesso a dados* são serviços Web que permitem a publicação de visões XML a partir de dados armazenados em uma fonte de dados.

Em nossa abordagem, a estratégia de utilizar *serviços Web de acesso a dados* oferece uma forma transparente e flexível para a publicação dados, como também encapsula detalhes de acesso e recuperação entre aplicações e dados. Outro fato importante é que o ***RelP*** utiliza a tradução de consultas para SQL/XML, o que torna a tradução eficiente, pois o resultado XML é processado pelo próprio SGBD.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1
Introdução

1.1 Descrição do Problema e Motivação

XML se estabeleceu como o formato padrão para a publicação, integração e troca de dados na Web. Esse sucesso se deve a sua capacidade de representar informações através de uma estrutura hierárquica simples e autodescritiva. Essa estrutura também facilita a exibição dos dados em páginas Web, devido à similaridade que existe entre XML e a estrutura em HTML usada para representar os mesmos dados. Outro fator importante para a popularização do XML é que as mensagens trocadas entre aplicações Web seguem especificações bem definidas, as quais geralmente possuem natureza hierárquica e podem facilmente ser implementadas usando XML.

Entretanto, a imensa maioria dos dados de negócio, incluindo dados de aplicações Web, continua sendo armazenada em SGBDs relacionais. Essa tendência tende a se manter, dada a segurança, escalabilidade, quantidade de ferramentas e performance associadas aos SGBDs relacionais [1, 2]. Atualmente, em muitos cenários, a mesma informação é armazenada e consultada em SGBDs relacionais, mas exibida e compartilhada no formato XML. Assim, muitas aplicações enviam dados XML a partir de dados relacionais e recebem dados XML que serão mapeados para uma estrutura de dados específica da aplicação ou para um banco relacional específico.

Representações relacionais e XML de um mesmo dado tendem a ser divergentes devido às diferenças entre esses modelos. Enquanto o modelo relacional é baseado em tabelas bidimensionais, as quais não possuem hierarquia ou ordem, o XML é baseado em árvores nas quais a ordem é significante. O modelo relacional se caracteriza por possuir atributos atômicos, por não existir a noção de ordem de atributos e por ser fortemente estruturado com definições estáticas de esquema, por exemplo, um mesmo esquema se aplica a todas as linhas de uma dada tabela. O modelo XML apresenta um conceito diferente, no qual a informação pode ser vista como uma árvore de nós (elementos), onde elementos podem conter outros elementos (aninhamento).

Além disso, pelas características do modelo semi-estruturado, dois documentos XML com árvores de nós diferentes podem apresentar o mesmo esquema.

Neste trabalho, tratamos o problema de publicação de dados relacionais no formato XML. Devido à larga utilização de aplicações Web que utilizam bancos de dados relacionais juntamente com a tecnologia XML, e ainda, devido às diferenças substanciais entre o modelo relacional e o modelo XML, este é um problema de extrema relevância e não trivial [7, 11, 16, 17, 18, 28].

De acordo com [13], um *framework* para publicação de dados relacionais no formato XML deve cumprir três requisitos:

(i) Deve ser **geral**, i.e., permitir diferentes formas de mapear dados relacionais em XML. Dados Relacionais são *flat*, normalizados e seu esquema geralmente não é público. Por exemplo, dados e atributos importantes podem ser omitidos aos usuários através de visões. Por outro lado, dados XML podem ser aninhados, não normalizados e seu DTD ou XML Schema é público. Dessa forma, o mapeamento de um modelo relacional para um modelo XML é complexo. Uma das limitações de muitos dos sistemas existentes é que um esquema relacional é mapeado em um esquema XML canônico (fixo). Isso não permite que o esquema de um banco de dados relacional possa ser mapeado em diferentes esquemas XML, para atender às diferentes aplicações.

(ii) Deve ser **seletivo**, i.e., deve ser capaz de materializar somente o fragmento do documento XML necessário para a aplicação. Assim, a visão XML deve ser virtual, de forma a permitir que as aplicações especifiquem através de consultas XML sobre a visão quais itens de dados são necessários, o que, em geral, é uma pequena parte do banco de dados.

(iii) Deve ser **eficiente**, i.e., deve garantir o processamento eficiente de consultas sobre as visões XML exportadas. Consultas definidas sobre a visão XML devem ser traduzidas em consultas sobre o banco de dados relacional, e o resultado deve ser retornado em XML. Consultas relacionais utilizam mecanismos de otimização sofisticados e eficientes. Um *framework* de publicação XML deve aproveitar-se desses mecanismos sempre que dados relacionais forem materializados no formato XML.

Diversos *middlewares* e *frameworks* foram propostos para solucionar o problema de publicação de dados relacionais no formato XML [7, 11, 16, 17, 18, 28]. No *XTables* [18] e *SilkRoute* [11], uma visão XML é definida como uma consulta XQuery sobre uma visão XML canônica, que representa o esquema relacional. Esta consulta especifica o esquema da visão e os mapeamentos que descrevem como o

esquema da visão está relacionado com a visão canônica. Uma consulta XQuery sobre a visão é traduzida em consultas SQL e os resultados SQL são rotulados (*tagged*) produzindo um documento XML. Nesses sistemas, o processamento eficiente de consulta não é garantido, pois dada uma consulta XQuery, uma ou mais consultas SQL podem ser geradas na tradução. Além disso, tais sistemas não apresentam formalismo de mapeamento; o mapeamento é feito através de consultas XQuery, as quais são difíceis de gerar e manter.

No *Framework de Três Fases* do Clio [7], uma visão é definida por um conjunto de *tuple-generating dependencies* (tgds) [10], que representam os mapeamentos. Estes mapeamentos podem ser gerados com ajuda da ferramenta Clio. O *framework* recebe como entrada os tgds e: (i) para cada tgd, os dados relevantes são extraídos da fonte de dados. Então, (ii) fragmentos XML são gerados a partir dos dados relevantes. Finalmente, (iii) os fragmentos XML são mesclados/agrupados para produzir o resultado final. O *framework* é utilizado para transformação e transporte de dados entre esquemas fonte-alvo.

No *XML Publisher* [17], sistema proposto anteriormente à nossa proposta, o ***RelP***, uma visão é definida por um esquema e um conjunto de Assertivas de Correspondências (ACs). Baseado nas ACs da visão, uma visão de objeto canônica é criada sobre o banco de dados. Uma consulta XQuery, definida sobre o esquema da visão, é primeiramente traduzida em uma consulta SQL:92 sobre o esquema da visão canônica e, através do mecanismo de visão do SGBD, a consulta é executada. A abordagem usada neste sistema é intrusiva, uma vez que a criação da visão de objeto canônica é obrigatória.

Atualmente, com a introdução do padrão SQL/XML [22], alguns SGBDs, como Oracle [3], DB2 [5] e postgreSQL [6], apresentam mecanismos que permitem ao usuário construir consultas declarativas, que retornam dados XML a partir de dados relacionais. O XML resultante da consulta pode ter a estrutura que o projetista desejar, e as consultas podem ser arbitrariamente complexas. Para um programador SQL, SQL/XML é fácil de aprender, pois envolve um grupo pequeno de adições à linguagem SQL existente. Uma vez que SQL é uma linguagem madura, existe um grande número de ferramentas e infra-estrutura disponíveis que podem ser facilmente adaptadas para suportar o novo padrão. Por exemplo, SQL/XML usa JDBC para retornar resultados. Outra vantagem da linguagem SQL/XML é que os mecanismos de otimização

desenvolvidos para o SQL/XML são baseados em otimizadores relacionais clássicos, o que representa um ganho considerável em desempenho [25].

Em resumo, de um lado, temos soluções baseadas em *middlewares* que apresentam linguagens de mapeamento complexas e que não garantem a eficiência no processamento de consultas. Ao mesmo tempo, temos poucos SGBDs que permitem a publicação de dados no formato XML usando o padrão SQL/XML, e é menor ainda o número de SGBDs que oferece um mecanismo de visão SQL/XML, i.e., uma visão XML que utiliza o SQL/XML. Além disso, a maioria desses SGBDs são proprietários e relativamente caros para os usuários. Dessa forma, a necessidade de soluções de baixo custo de aquisição, que ofereçam suporte a criação de visões XML de forma simples, que garantam acesso fácil aos dados e processamento eficiente de consultas sobre as visões, motivou este trabalho.

1.2 Objetivos e Contribuições

Neste trabalho, apresentamos um *middleware*, chamado ***RelP***, para publicação e consulta de dados relacionais através de visões SQL/XML. No ***RelP***, um usuário pode realizar, via *serviço Web de acesso a dados*, consultas sobre uma visão SQL/XML. A consulta é traduzida em uma consulta SQL/XML equivalente definida sobre o esquema do banco de dados relacional. *Serviços Web de acesso a dados* são serviços Web que permitem a publicação de visões XML a partir de dados armazenados em uma fonte de dados.

Em nossa abordagem, a estratégia de utilizar *serviços Web de acesso a dados* oferece uma forma transparente e flexível para a publicação dados, como também encapsula detalhes de acesso e recuperação entre aplicações e dados. Outro fato importante é que o ***RelP*** traduz consultas XQuery sobre o esquema da visão em consultas SQL/XML sobe o banco, o que torna a tradução eficiente, pois o resultado XML é processado pelo próprio SGBD.

Neste trabalho, uma visão XML é especificada por um esquema XML e um conjunto de Assertivas de Correspondência (ACs), as quais definem o mapeamento entre o esquema da visão e o esquema do banco relacional. As visões devem estar de

acordo com um esquema XML pré-definido (*schema-directed publishing*). Deste modo, as Assertivas de Correspondência induzem mapeamentos definidos pela classe de consultas SQL/XML da forma *projeção-seleção-equijunção* (PSE). Essa classe de consultas oferece suporte à maioria dos tipos de reestruturação de dados que são comuns em aplicações de troca de dados. Tais restrições à expressividade dos mapeamentos permitem garantir uma melhor eficiência no processamento de consultas, como veremos no Capítulo 7.

A seguir, descrevemos as principais contribuições deste trabalho:

1. Especificação de uma interface padrão de acesso a visões XML via Web.

A especificação *Web XML Service* (WXS) provê uma interface padrão para a publicação e consultas de visões XML na Web. A especificação define o conjunto de métodos, descritos a seguir:

- getCapabilities, permite aos usuários solicitar e recuperar a descrição dos metadados ou arquivo de capacidades, que são os documentos que descrevem as habilidades da implementação específica do servidor;
- getViewType, permite ao usuário recuperar a descrição da estrutura de qualquer visão XML (um esquema XML);
- query, permite ao usuário recuperar instâncias de uma visão XML usando a linguagem XQuery.

2. Proposta e implementação do *middleware RelP*.

A Figura 1.1 mostra os principais componentes e ferramentas do ***RelP***, os quais são descritos a seguir.

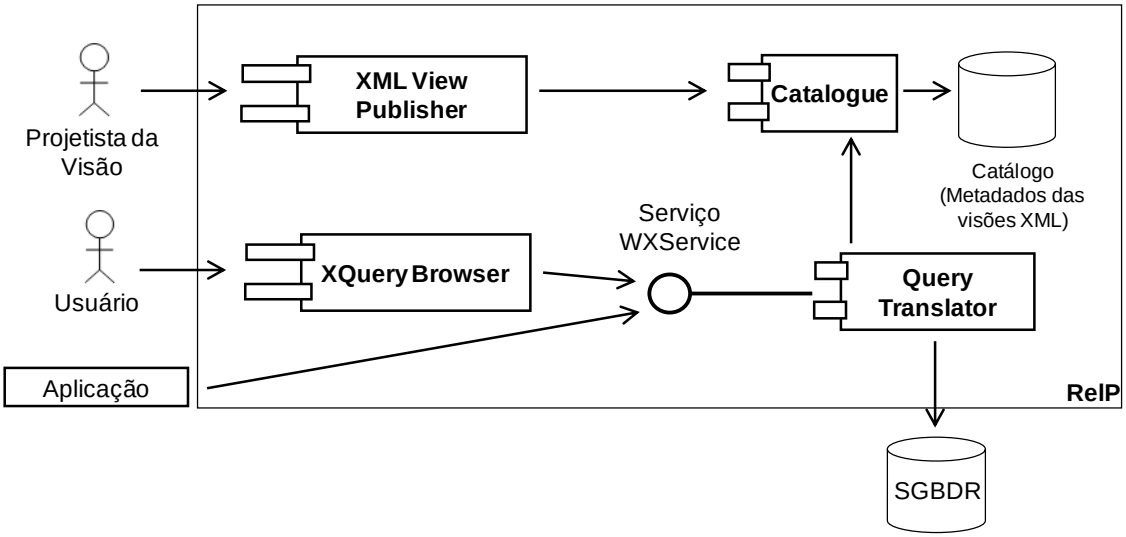


Figura 1.1: Componentes e ferramentas do ***RelP***.

i) A Ferramenta ***XML View Publisher*** (XVP) suporta a criação, registro e manutenção de visões XML. Através da interface gráfica da ferramenta XVP, os usuários podem criar o esquema da visão e suas Assertivas de Correspondência (ACs) de forma intuitiva, sem a necessidade de ter o conhecimento profundo do formalismo das ACs. Com base nas Assertivas de correspondência da visão, a ferramenta XVP gera automaticamente os Templates de Consulta, os quais são constituídos de fragmentos de consultas SQL/XML, sendo estes usados pelo ***Query Translator*** para traduzir uma consulta XQuery em uma consulta SQL/XML.

ii) *O Catalogue* é o componente responsável pelo armazenamento dos metadados das visões XML (esquema XML e Templates de Consulta da visão) na base Catálogo;

iii) *O Query Translator* é o componente de software responsável pela tradução de uma consulta XQuery definida sobre o esquema de uma visão em uma consulta SQL/XML sobre o esquema do banco relacional. A tradução é feita com o auxílio dos *Templates de Consulta* (TCs) da visão. Isso simplifica a tradução de consulta, uma vez que todo o processamento dos mapeamentos complexos da visão é realizado em tempo de projeto. Deste modo, como veremos no Capítulo 5, a reescrita da consulta se resume a uma composição dos fragmentos relevantes para construir a consulta SQL/XML correta.

iv) A Ferramenta *XQuery Browser* permite ao usuário que não domina a linguagem XQuery construir intuitivamente de forma gráfica consultas sobre as visões XML publicadas.

v) O serviço *WXService* é um serviço Web que implementa a especificação *Web XML Service*. *WXService* implementa os seguintes métodos da especificação: getCapabilities, getViewType e query:

3. Adaptação do *middleware RelP* à especificação *Web Feature Service* do OGC.

Na área geográfica, devido ao alto custo de produção e complexidade dos dados espaciais, o compartilhamento de informações tem sido incentivado através do uso de *serviços Web de Acesso a Dados* geográficos. Neste contexto, o Consórcio OpenGIS (OGC) [35] promoveu duas importantes iniciativas: a linguagem *Geography Markup Language* (GML) [36], padrão para a publicação de dados geográficos, e a especificação *Web Feature Service* (WFS) [38], formato padrão para acesso e manipulação de dados GML na Web.

Com base nessas iniciativas, propomos uma adaptação do nosso *middleware* à especificação WFS, de modo que seja possível publicar e consultar visões GML sobre base de dados relacional. Para tanto, implementamos um outro serviço, chamado *WFService*, o qual é uma adaptação do serviço *WXService* com base na especificação WFS. O serviço *WFService* implementa os seguintes métodos da especificação: getCapabilities, para recuperar o arquivo de capacidades do servidor e as visões GML publicadas; describeFeatureType, para recuperar a descrição da estrutura de qualquer visão GML; e getFeature, para recuperar instâncias de visões GML através de consultas WFS. Além disso, adaptamos o algoritmo de tradução de consultas XQuery do serviço *WXService* para traduzir consultas WFS sobre visões GML em consultas SQL/XML sobre o banco de dados. Foi adaptado também, a ferramenta *XML View Publisher*, para tratar visões GML, e a ferramenta *XQuery Browser*, para criar consultas WFS de forma gráfica.

1.3 Organização da Dissertação

Essa dissertação está organizada como apresentado a seguir. O Capítulo 2 apresenta o padrão SQL/XML, o qual é usado para consultar dados XML a partir de dados relacionais. O Capítulo 3 descreve brevemente a tecnologia dos serviços Web e apresenta a especificação *Web XML Service* (WXS). O Capítulo 4 apresenta o processo de publicação de visões XML no ***RelP***. O Capítulo 5 mostra o processo de tradução de consultas XQuery no ***RelP***. O Capítulo 6 apresenta a adaptação do ***RelP*** à especificação WFS, mostrando o processo e os algoritmos de publicação de visões GML e de tradução consultas WFS. O Capítulo 7 descreve os trabalhos relacionados mais recentes e apresenta uma análise da eficiência do processamento de consultas do ***RelP***. Finalmente, o Capítulo 8 apresenta as conclusões e sugestões para trabalhos futuros.

Capítulo 2
SQL/XML

Neste Capítulo, apresentamos o padrão SQL/XML, o qual é usado para consultar dados XML a partir de dados relacionais. O Capítulo está organizado da seguinte forma. A Seção 2.1 discute detalhes do padrão SQL/XML e a Seção 2.2 discute as vantagens do uso da SQL/XML para criar visões XML de dados relacionais.

2.1 O Padrão SQL/XML

Atualmente, grande parte dos SGBDs possui extensões proprietárias para o gerenciamento de dados XML, como: Oracle, PostgreSQL, DB2, MySQL, etc. Essas extensões usam diferentes abordagens, obrigando os usuários que trabalham com diversos bancos a escreverem códigos diferentes, um para cada SGBD.

SQL/XML [22] é um padrão ANSI/ISO desenvolvido pelo INCITS H2.3 que estende o SQL. Com SQL/XML é possível construir consultas declarativas que retornam dados XML a partir de bases relacionais. De fato, alguns SGBDs já apresentam mecanismos para a publicação de dados XML através de consultas SQL/XML [3, 5, 6]. A vantagem desta abordagem é que os mecanismos de otimização desenvolvidos para o SQL/XML são baseados nos mecanismos de otimizadores relacionais clássicos, o que representa um ganho considerável em desempenho [25].

Uma vez que SQL é uma linguagem madura, existe um grande número de ferramentas e infra-estrutura disponíveis que podem ser facilmente adaptadas para suportar o padrão SQL/XML. Além disso, SQL/XML é fácil de aprender, pois envolve um grupo pequeno de adições à linguagem SQL existente.

No padrão SQL/XML, as funções de publicação constituem a parte que provê a publicação de dados relacionais no formato XML. As funções de publicação são usadas diretamente na consulta SQL e permitem criar qualquer estrutura XML que o usuário deseje a partir do esquema relacional. O resultado de uma função de publicação é sempre uma instância do tipo de dados XML, o qual também é definido pelo padrão.

As principais funções de publicação do SQL/XML são: XMLElement(), XMLAttributes(), XMLForest(), XMLAgg() e XMLConcat().

A seguir é descrito detalhadamente as funções de publicação. Nesta seção, considere o esquema relacional Pedidos_BD descrito na Figura 2.1 e um estado de Pedido_BD apresentado na Figura 2.2

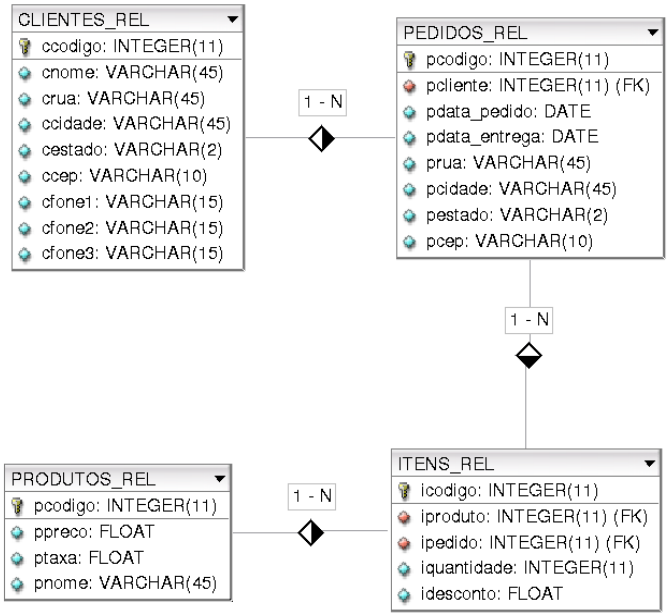


Figura 2.1: Esquema relacional Pedidos_BD.

CLIENTES_REL

CCODIGO	CNOME	CRUA	CCIDADE	CESTADO	CCEP	CFONE1	CFONE2	CFONE3
1010	Narah Jones	8 Abolicao II	Fortaleza	CE	12205	0959 0432	0838 5454	0162 3947
1020	Adam Smith	400 W. Soares	Natal	RN	21286	0999 0000	0183 4813	0012 4827
1020	Mary Jane	122 R. Alencar	Salvador	BA	65478	0012 4825	NULL	NULL

PRODUTOS_REL

PCODIGO	PNOME	PPRECO	PTAXA
2638	Monitor	400.00	0.01
1721	Teclado	25.00	0.005
1761	Mouse	16.90	0.0

ITENS_REL

IPEDIDO	ICODIGO	IPRODUTO	IQUANTIDADE	IDESCONTO
300	1	2638	35	0.07
300	1	1721	15	0.07
310	2	1721	30	0.07
320	3	1761	20	0.05
320	3	2638	25	0.05
540	4	1761	60	0.10

PEDIDOS_REL

PCODIGO	PCLIENTE	PDATA_PEDIDO	PDATA_ENTREGA	PRUA	PCIDADE	PESTADO	PCEP
300	1010	01/07/09	05/07/09	8 Abolicao II	Fortaleza	CE	12205
310	1020	10/07/09	20/07/09	400 W. Soares	Natal	RN	21286
320	1030	12/07/09	13/07/09	122 R. Alencar	Salvador	BA	65478
540	1010	15/10/09	20/10/09	8 Abolicao II	Fortaleza	CE	12205

Figura 2.2: Um estado de Pedidos_BD.

XMLElement e **XMLAttributes** – A função XMLElement constrói um novo elemento XML com atributos e conteúdo. A função recebe como parâmetro o nome do elemento, um conjunto opcional de atributos para o elemento, e zero ou mais argumentos adicionais que compõem o conteúdo do elemento. O resultado da função é uma instância do tipo XMLType. Caso todos os argumentos da função retornem valor nulo, então nenhum conteúdo é gerado para o elemento.

A função XMLAttributes especifica os atributos do elemento raiz. A função recebe como argumento um ou mais expressões escalares que podem apresentar *aliases* ou não. Se o valor de alguma expressão for nulo, o atributo correspondente não é gerado. A função XMLAttributes só pode ser chamada como primeiro argumento da função XMLElement.

Como exemplo, considere a seguinte consulta sobre uma relação CLIENTES_REL, apresentada na Figura 2.1.

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLElement("fone", C.CFONE1),
          XMLElement("fone", C.CFONE2),
          XMLElement("fone", C.CFONE3) )
FROM CLIENTES_REL C;
```

O resultado da consulta é dado por:

```
<cliente codigo="1010">
   <nome>Narah Jones</nome>
   <fone>0959 0432</fone>
   <fone>0838 5454</fone>
   <fone>0162 3947</fone>
</cliente>
<cliente codigo="1020">
   <nome>Adam Smith</nome>
   <fone>0999 0000</fone>
   <fone>0183 4813</fone>
   <fone>0012 4827</fone>
</cliente>
<cliente codigo="1030">
   <nome>Mary Jane</nome>
   <fone>0012 4835</fone>
   <fone></fone>
   <fone></fone>
</cliente>
```

O identificador cliente dá nome ao elemento raiz. A expressão C.CCODIGO AS "codigo" dentro da função XMLAttributes gera o atributo codigo do elemento a partir do valor da expressão escalar C.CCODIGO. As expressões XMLElement("nome", C.CNOME),

XMLElement("fone", C.CFONE1), XMLElement("fone", C.CFONE2) e XMLElement("fone", C.CFONE3) geram o conteúdo do elemento <cliente>. Em XMLElement("nome", C.CNOME), o valor escalar da expressão C.CNOME é mapeado no valor XML equivalente, gerando assim o conteúdo do elemento <nome>. Note que, no terceiro elemento <cliente> do resultado, como os valores de C.CFONE2 e C.CFONE3 são nulos, então nenhum conteúdo é gerado para o elemento correspondente.

XMLForest – A função XMLForest gera uma floresta de elementos XML a partir de seus argumentos, os quais são expressões escalares com *aliases* opcionais. Cada expressão é convertida no formato XML correspondente e, caso um *alias* tenha sido atribuído, este será o identificador do elemento gerado. Diferente da função XMLElement, XMLForest pode somente definir uma sequência de elementos XML e expressões que resultam em nulo não geram elementos.

Como exemplo, considere a seguinte consulta sobre a relação CLIENTES_REL:

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone") )
FROM CLIENTES_REL C;
```

O resultado da consulta é dado por:

```
<cliente codigo="1010">
   <nome>Narah Jones</nome>
   <fone>0959 0432</fone>
   <fone>0838 5454</fone>
   <fone>0162 3947</fone>
</cliente>
<cliente codigo="1020">
   <nome>Adam Smith</nome>
   <fone>0999 0000</fone>
   <fone>0183 4813</fone>
   <fone>0012 4827</fone>
</cliente>
<cliente codigo="1030">
   <nome>Mary Jane</nome>
   <fone>0012 4835</fone>
</cliente>
```

O identificador fone dá nome aos elementos gerados das expressões C.CFONE1 AS "fone", C.CFONE2 AS "fone" e C.CFONE3 AS "fone". Note que, no terceiro elemento <cliente> do resultado, como os valores de C.CFONE2 e C.CFONE3 são nulos, então nenhum elemento foi gerado.

XMLAgg – É uma função de agregação que gera uma agregado XML (uma lista de elementos XML) a partir de cada agrupamento SQL. Se nenhum agrupamento for especificado, é retornado um agregado XML para todas as cláusulas da consulta. Argumentos nulos são removidos do resultado. A função recebe como parâmetro uma única expressão que gera uma instância do tipo XMLType.

Como exemplo, considere as seguintes consultas sobre a relação CLIENTES_REL:

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone"),
          (SELECT XMLAgg( XMLElement("pedido",
                     XMLElement("codigo", P.PCODIGO),
                     XMLElement("data", P.PDATA_PEDIDO) ) )
           FROM PEDIDOS_REL P
           WHERE P.PCLIENTE = C.CCODIGO ) )
FROM CLIENTES_REL C;
```

e

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone"),
          XMLAgg(
             XMLElement("pedido",
                XMLElement("codigo", P.PCODIGO),
                XMLElement("data", P.PDATA_PEDIDO) ) ) )
FROM CLIENTES_REL C, PEDIDOS_REL P
WHERE P.PCLIENTE = C.CCODIGO
GROUP BY C.CCODIGO, C.CNOME, C.CFONE1, C.CFONE2, C.CFONE3,
         P.PCODIGO, P.PDATA_PEDIDO;
```

O resultado de ambas é o mesmo, dado por:

```
<cliente codigo="1010">
   <nome>Narah Jones</nome>
   <fone>0959 0432</fone>
   <fone>0838 5454</fone>
   <fone>0162 3947</fone>
```

```
   <pedido>
      <codigo>300</codigo>
      <data>01/07/09</data>
   </pedido>
   <pedido>
      <codigo>540</codigo>
      <data>15/10/09</data>
   </pedido>
</cliente>
<cliente codigo="1020">
   <nome>Adam Smith</nome>
   <fone>0999 0000</fone>
   <fone>0183 4813</fone>
   <fone>0012 4827</fone>
   <pedido>
      <codigo>310</codigo>
      <data>10/07/09</data>
   </pedido>
</cliente>
<cliente codigo="1030">
   <nome>Mary Jane</nome>
   <fone>0012 4835</fone>
   <pedido>
      <codigo>320</codigo>
      <data>12/07/09</data>
   </pedido>
</cliente>
```

Na primeira consulta, é retornado um agregado XML para todas as cláusulas da subconsulta. Na segunda consulta, é retornado um agregado XML para cada agrupamento definido pela cláusula GROUP BY.

XMLConcat – A função XMLConcat concatena dois ou mais valores XML (expressão de tipo de dados XML). Esses valores são passados como parâmetro para a função, a qual gera uma instância do tipo XMLType. Valores nulos são removidos do resultado.

Como exemplo, considere a seguinte consulta sobre a relação PRODUTOS_REL:

```
SELECT XMLElement("produto",
           XMLConcat(
                XMLElement("nome", PROD.PNOME),
                XMLElement("preco", PROD.PPRECO) ) )
FROM PRODUTOS_REL PROD
```

O resultado da consulta é dado por:

```
<produto>
   <nome>Monitor</nome>
   <preco>400,00</preco>
</produto>
<produto>
```

```
   <nome>Teclado</nome>
   <preco>25,00</preco>
</produto>
<produto>
   <nome>Mouse</nome>
   <preco>16,90</preco>
</produto>
```

Note que a consulta está concatenando os valores XMLElement("nome", PROD.PNOME) e XMLElement("preco", PROD.PPRECO), e gerando uma única instância XML desses valores, contidas nos elementos <produto> do resultado.

2.2 Definição de visões XML

O problema de publicar dados relacionais no formato de XML tem especial importância, uma vez que XML emergiu como o formato padrão para troca de dados entre aplicações Web enquanto a maioria dos dados de negócio continua armazenada em SGBDs relacionais. Uma maneira geral e flexível de publicar dados relacionais no formato XML é através da criação de visões XML sobre dados relacionais [2][54]. Essa estratégia permite aos usuários consultar sobre visões sem a necessidade de entender como o banco de dados está estruturado. Além disso, a especificação de determinadas consultas sobre as visões podem se tornar mais simples.

A introdução do padrão SQL/XML em SGBDs permite aos usuários criar uma visão de instâncias de um tipo XML sobre tabelas relacionais usando as funções de publicação do padrão SQL/XML, como XMLElement(), XMLConcat(), etc. Atualmente, os SGBDs Oracle, DB2 e PostgreSQL são os únicos que suportam a criação de visões XML utilizando as funções de publicação do SQL/XML.

No Oracle, uma visão XML é uma visão com instâncias do tipo XMLType, as quais podem existir fisicamente no banco de dados (em tabelas XML) e podem ser "sintetizados" a partir de dados no formato relacional.

Exemplo 2.1

Considere, por exemplo, o esquema relacional Pedidos_BD e o esquema da visão XML Pedidos_XML, mostradas nas Figuras 2.1 e 2.3, respectivamente. A visão Pedidos_XML é um conjunto de elementos <Pedido>, os quais são instâncias do tipo TPedido_XML. Para gerar instâncias de TPedido_XML a partir de Pedidos_BD, definimos a visão SQL/XML no Oracle mostrada na Figura 2.4. Como ilustrado nas Figuras 2.2 e 2.5, para cada tupla da tabela PEDIDOS_REL, a visão XML usa as funções de publicação do padrão SQL/XML para construir uma instância do tipo XML TPedido_XML.

Mais detalhadamente, considere o estado da base de dados Pedidos_DB mostrado na Figura 2.2. O estado correspondente de Pedidos_XML é mostrado na Figura 2.5. Esta instância da visão contém uma seqüência de elementos <Pedido> do tipo TPedido_XML, que são os elementos primários da visão. Cada elemento <Pedido> é construído a partir de uma tupla da tabela de PEDIDOS_REL usando a função de publicação XMLElement() do SQL/XML. Os subelementos e atributos de <Pedido> são construídos através de subconsultas SQL/XML. Por exemplo, o atributo ID é construído usando o subconsulta na linha 4, o subelemento <DataPedido> é construído usando o subconsulta na linha 5, o subelemento <Item> é construído pela subconsulta nas linhas 18-29. Nas linhas 18 – 29, temos que, para cada tupla na tabela PEDIDOS_REL, as tuplas relevantes da tabela ITENS_REL são recuperadas e convertidas em uma seqüência de elementos <Item>.

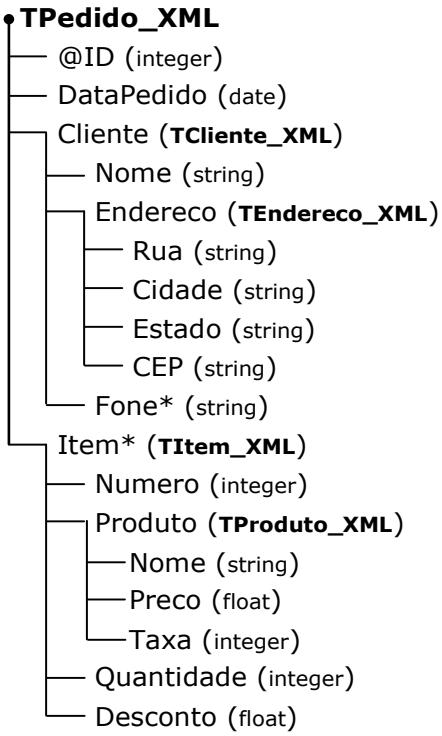


Figura 2.3: Esquema da visão XML Pedidos_XML.

```
1.  CREATE OR REPLACE VIEW PEDIDOS_XML OF XMLTYPE
2.  XMLSCHEMA "Pedido.xsd" ELEMENT "Pedido"
3.  AS SELECT XMLELEMENT("Pedido",
4.      XMLATTRIBUTES(P.PCODIGO AS "ID"),                                  TPedido_XML]
5.      XMLFOREST(P.PDATA_ENTREGA AS "DataPedido"),
6.      (SELECT XMLELEMENT("Cliente",
7.          XMLFOREST(C.CNOME AS "Nome"),                                  TCliente_XML]
8.          XMLELEMENT("Endereco",
9.              XMLFOREST(C.CRUA AS "Rua"),                                TEndereco_XML]
10.             XMLFOREST(C.CCIDADE AS "Cidade"),
11.             XMLFOREST(C.CESTADO AS "Estado"),
12.             XMLFOREST(C.CCEP AS "CEP") ),
13.         XMLFOREST(C.CFONE1 AS "Fone"),
14.         XMLFOREST(C.CFONE2 AS "Fone"),
15.         XMLFOREST(C.CFONE3 AS "Fone") )
16.     FROM CLIENTES_REL C
17.     WHERE C.CCODIGO = P.PCLIENTE),
18.     (SELECT XMLAGG( XMLELEMENT("Item",
19.         XMLFOREST(I.ICODIGO AS "Numero"),                              TItem_XML]
20.         (SELECT XMLELEMENT("Produto",
21.             XMLFOREST(D.PNOME AS "Nome"),                              TProduto_XML]
22.             XMLFOREST(D.PPRECO AS "Preco"),
23.             XMLFOREST(D.PTAXA AS "Taxa") )
24.         FROM PRODUTOS_REL D
25.         WHERE D.PCODIGO = I.IPRODUTO),
26.         XMLFOREST(I.IQUANTIDADE AS "Quantidade"),
27.         XMLFOREST(I.IDESCONTO AS "Desconto") ) )
28.     FROM ITENS_REL I
29.     WHERE I.IPEDIDO = P.PCODIGO ) )
30. FROM PEDIDOS_REL P
```

Figura 2.4: Definição SQL/XML da visão Pedidos_XML.

```
<Pedido ID="300">
  <DataPedido>2009-07-05</DataPedido>
  <Cliente>
    <Nome>Narah Jones</Nome>
    <Endereco>
       <Rua>8 Abolicao II</Rua>
       <Cidade>Fortaleza</Cidade>
       <Estado>CE</Estado>
       <CEP>12205</CEP>
    </Endereco>
    <Fone>0959 0432</Fone>
    <Fone>0838 5454</Fone>
    <Fone>0162 3947</Fone>
  </Cliente>
  <Item>
     <Numero>1</Numero>
     <Produto>
        <Nome>Monitor</Nome>
        <Preco>400,00</Preco>
        <Taxa>0.01</Taxa>
     </Produto>
     <Quantidade>35</Quantidade>
     <Desconto>0.07</Desconto>
   </Item>
  <Item>
     <Numero>1</Numero>
     <Produto>
        <Nome>Teclado</Nome>
        <Preco>25,00</Preco>
        <Taxa>0.005</Taxa>
     </Produto>
     <Quantidade>15</Quantidade>
     <Desconto>0.07</Desconto>
  </Item>
</Pedido>
```

```
<Pedido ID="310">
  <DataPedido>2009-07-20</DataPedido>
  <Cliente>
     <Nome>Adam Smith</Nome>
     <Endereco>
        <Rua>400 W. Soares</Rua>
        <Cidade>Natal</Cidade>
        <Estado>RN</Estado>
        <CEP>21286</CEP>
     </Endereco>
     <Fone>0999 0000</Fone>
     <Fone>0183 4813</Fone>
     <Fone>0012 4827</Fone>
  </Cliente>
  <Item>
     <Numero>2</Numero>
     <Produto>
        <Nome>PC Bag - L/S</Nome>
        <Preco>256,28</Preco>
        <Taxa>0.005</Taxa>
     </Produto>
     <Quantidade>30</Quantidade>
     <Desconto>0.05</Desconto>
  </Item>
</Pedido>
```

Figura 2.5: Um estado da visão Pedidos_XML.

Para consultar visões XML, os SGBDs Oracle e DB2 permitem aos usuários utilizar a linguagem XQuery. Assim, usuários que não conhecem as funções de publicação da linguagem SQL/XML podem consultar dados XML a partir de dados relacionais definindo consultas XQuery sobre uma visão SQL/XML. Além disso, os usuários se beneficiam da eficiência e segurança oferecidas pelo padrão SQL/XML.

No Oracle, a função XMLQuery [24] permite realizar consultas XQuery sobre visões XML. Para executar as consultas XQuery, o Oracle faz a tradução da mesma em uma consulta SQL/XML equivalente. Basicamente, a tradução é realizada em três passos: (i) a consulta XQuery é traduzida em uma representação XML XQueryX. Em seguida, (ii) a representação XQueryX é transformada em uma árvore, chamada XQuery Expression Tree (XET). Em seguida, (iii) a árvore é analisada semanticamente construindo a consulta SQL/XML. O processo completo de reescrita de uma consulta XQuery em uma consulta SQL/XML no Oracle é descrito em [24, 25].

Exemplo 2.2

Considere por exemplo, a consulta SQL Xq apresentada na Figura 2.6 sobre a visão Pedidos_XML. Xq é uma consulta SQL com uma consulta XQuery embutida na função XMLQuery. Xq retorna todos os clientes dos pedidos cujo id é maior que 100. A Figura 2.7 apresenta a consulta SQL/XML Sq gerada a partir de Xq.

Considere o estado de Pedidos_BD apresentado na Figura 2.2. O resultado da consulta Sq com instâncias do tipo da visão TPedido_XML é apresentado na Figura 2.8.

```
SELECT
   XMLQuery(
      ' FOR $i in ./Pedido
        WHERE $i/@id > 100
       RETURN $i/Cliente'
   RETURNING CONTENT)
FROM Pedidos_XML
```

Figura 2.6: Consulta XQuery Xq.

```
SELECT XMLELEMENT("Pedido",
(SELECT XMLELEMENT("Cliente",
            XMLFOREST(C.CNOME AS "NOme"),
            XMLELEMENT("Endereco",
                  XMLFOREST(C.CRUA AS "Rua"),
                  XMLFOREST(C.CCIDADE AS "Cidade"),
                  XMLFOREST(C.CESTADO AS "Estado"),
                  XMLFOREST(C.CCEP AS "CEP") ),
            XMLFOREST(C.CFONE1 AS "Fone"),
            XMLFOREST(C.CFONE2 AS "Fone"),
            XMLFOREST(C.CFONE3 AS "Fone") )
      FROM CLIENTES_REL C
      WHERE C.CCODIGO = P.PCLIENTE),
FROM PEDIDOS_REL P
WHERE P.PCODIGO > 100
```

Figura 2.7: Consulta SQL/XML Sq.

```
<Pedido>
   <Cliente>
    <Nome>Narah Jones</Nome>
    <Endereco>
        <Rua>8 Abolicao II</Rua>
        <Cidade>Fortaleza</Cidade>
        <Estado>CE</Estado>
        <CEP>12205</CEP>
    </Endereco>
    <Fone>0959 0432</Fone>
    <Fone>0838 5454</Fone>
    <Fone>0162 3947</Fone>
  </Cliente>
</Pedido>
<Pedido>
  <Cliente>
     <Nome>Adam Smith</Nome>
     <Endereco>
        <Rua>400 W. Soares</Rua>
        <Cidade>Natal</Cidade>
        <Estado>RN</Estado>
        <CEP>21286</CEP>
     </Endereco>
     <Fone>0999 0000</Fone>
     <Fone>0183 4813</Fone>
     <Fone>0012 4827</Fone>
  </Cliente>
</Pedido>
.....
```

Figura 2.8: Estado resultante da consulta Sq.

Capítulo 3
Serviços Web de Acesso a Dados

Neste Capítulo, apresentamos os serviços Web e a especificação Web XML Service, a qual provê uma interface padrão para a publicação e consultas de visões XML na Web. O Capítulo está organizado da seguinte forma. A Seção 3.1 discute sobre serviços Web e Serviços Web de Acesso a Dados, e a Seção 3.2 apresenta a especificação Web XML Service (WXS) com sua definição e descrição de seus métodos: getCapabilities, getViewType e query.

3.1 Serviços Web

Serviços *Web* ou *Web services*, constituem uma classe particular de serviços que usam padrões abertos adotados na *internet*, como o *Hypertext Transfer Protocol* (HTTP) para conexão e comunicação, *Uniform Resource Identifier* (URI) para identificação, *eXtensible Markup Language* (XML) para especificação de conteúdo, *Web Services Description Language* (WSDL) para descrição de serviço, e *Universal Description, Discovery and Integration* (UDDI) para implementar serviços de diretório. Assim, enquanto os serviços em geral provêem interoperabilidade entre diferentes componentes de *software*, os serviços *Web* avançam um passo ao facilitar o intercâmbio interinstitucional de dados e serviços sobre a *internet*, provendo o compartilhamento de recursos em ambientes heterogêneos, independentemente de plataforma ou linguagem de programação. Deste modo, novas aplicações podem interagir com aplicações já existentes, e sistemas desenvolvidos em diferentes plataformas e sistemas operacionais podem se tornar compatíveis.

Como mostra a Figura 3.1, os serviços Web apresentam uma arquitetura baseada em entidades. As entidades correspondem aos papéis que os serviços Web podem assumir:

- Consumidor: representa o usuário ou a aplicação que solicita serviços;
- Provedor de Serviços: responsável por publicar serviços na Web;

- Catálogo: responsável por armazenar informações dos serviços publicados, como descrição, localização e métodos de acesso.

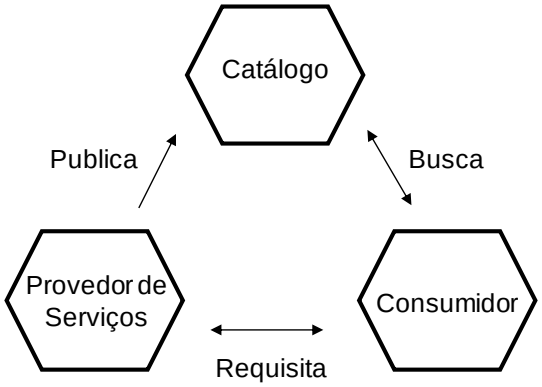


Figura 3.1: Arquitetura baseada em entidades dos serviços Web.

De acordo com a arquitetura, as principais operações que envolvem os serviços Web são: publicação, busca e requisição (chamada) de serviços. A operação *publicação de serviços* é realizada pelo provedor do serviço, que consiste na criação de uma descrição do serviço e publicação em canais de localização ou de descoberta na Web. A operação *busca de serviços* é realizada pelo consumidor, quando este procura por serviços Web no Catálogo. A operação *requisição de serviços* é o processo de comunicação consumidor-provedor, o qual se caracteriza quando o consumidor utiliza o serviço através de trocas de mensagens com o provedor.

As vantagens dessa arquitetura [63] são: interoperabilidade a baixo custo, principalmente por não assumir nenhum detalhe específico sobre a implementação das aplicações clientes; redução da complexidade por meio do encapsulamento de interfaces e componentes; e ainda, prolongamento da utilização de aplicações legadas. Devido a essas vantagens, os serviços Web têm sido amplamente utilizados como mecanismo interoperável de extração e acesso a dados armazenados em fontes distintas. Tais serviços são denominados *serviços Web de Acesso a Dados*. Eles permitem a publicação de visões a partir de dados armazenados em uma fonte de dados. Esta estratégia não somente oferece uma forma transparente e flexível para publicar dados, como também encapsula detalhes de acesso e recuperação entre aplicações e dados.

Na área geográfica, em que o compartilhamento de informações é importante devido ao alto custo de produção e a complexidade dos dados espaciais, os *serviços Web de Acesso a Dados* são bastante adotados. O *Open Geospatial Consortium*

(OGC) [35] define as recomendações de padronização que garantem a interoperabilidade entre Sistemas de Informações Geográficas[1] (SIGs) na Web através de *serviços Web de Acesso a Dados*. Duas importantes iniciativas do OGC são: a *Geography Markup Language* (GML) [36] e a especificação *Web Feature Service* (WFS) [38]. A GML é uma linguagem baseada em XML, sendo considerada o formato padrão para representação de dados geográficos na Web. A especificação WFS, por sua vez, tem o propósito de descrever as operações (métodos) WFS de manipulação de dados codificados em GML.

Nesse contexto, foi verificado que não existem entidades que especifiquem padrões para descrever operações de manipulação de dados em XML. Atualmente, as aplicações Web que manipulam dados em XML não apresentam uma interface padrão de acesso a esses dados, o que torna difícil para os usuários e para as aplicações obterem uma visualização genérica dos dados. Dessa forma, propomos a especificação *Web XML Service* (WXS), a qual provê uma interface padrão para acesso a dados relacionais no formato XML na Web.

3.2 Especificação Web XML Service

A especificação WXS provê uma interface padrão, como ilustrado na Figura 3.2, na qual o usuário ou serviço Web pode publicar e consultar visões XML a partir de dados armazenados em uma fonte de dados.

A especificação WXS é baseada na especificação WFS e define os métodos: (i) getCapabilities, permite aos usuários solicitar e recuperar a descrição dos metadados ou arquivo de capacidades, que são os documentos que descrevem as habilidades da implementação específica do servidor; (ii) getViewType, permite ao usuário recuperar a descrição da estrutura de qualquer visão XML (um esquema XML); e (iii) query, permite ao usuário recuperar instâncias de uma visão XML usando a linguagem XQuery.

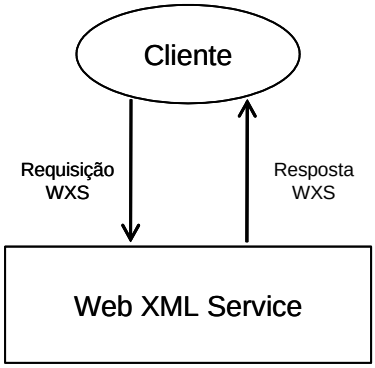


Figura 3.2: *Web XML Service*.

3.2.1 Método getCapabilities

O método getCapabilities descreve as capacidades do servidor WXS. Em outras palavras, este método deve indicar, principalmente, quais visões XML estão publicadas.

A Figura 3.3 mostra o *XML Schema* de uma requisição getCapabilities, a qual é composta do elemento getCapabilities e do atributo service informando a descrição do serviço.

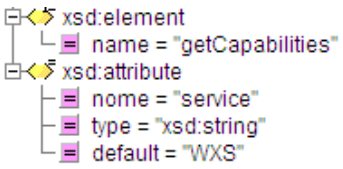


Figura 3.3: *XML Schema* da requisição getCapabilities.

As Figuras 3.4(a) e (b) mostram um exemplo de requisições getCapabilities usando o método HTTP POST (formato XML) e HTTP GET, respectivamente. Para a tecnologia SOAP, coloca-se a requisição getCapabilities que está no formato XML no conteúdo do elemento <Body> da mensagem.

```
<?xml version=‘ 1.0’?>
<GetCapabilities
        service=  ‘WXS’
        <!–- namespaces ->  />
```

(a)

```
http://www. ip_do_servidor_wxs:porta/
SERVICE=WXS&
REQUEST=GetCapabilities
```

(b)

Figura 3.4: (a) Exemplo da requisição getCapabilities utilizando HTTP POST. (b) Exemplo da requisição getCapabilities utilizando o método HTTP GET.

A Figura 3.5 mostra o *XML Schema* do documento XML resposta a uma requisição getCapabilities, o qual é composto do elemento raiz WXS_Capabilities seguido pela seqüência dos subelementos: Service, que apresenta meta-informações sobre o próprio serviço WXS, permitindo que este se auto-descreva; Capability, que contém meta-informações sobre requisições que o servidor WXS suporta; e XMLViewsList, que descreve uma lista de visões XML publicadas no servidor WXS.

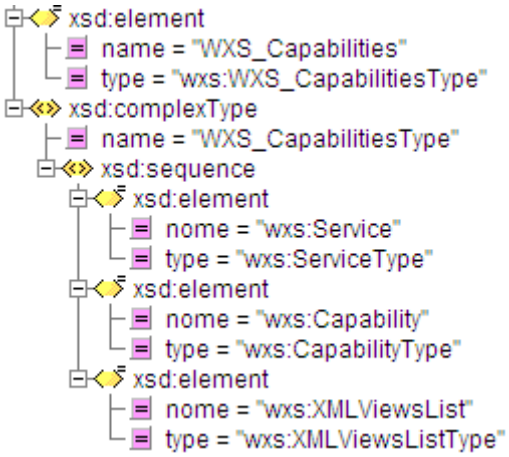


Figura 3.5: *XML Schema* resposta da requisição getCapabilities.

O *XML Schema* da Figura 3.6(a) define o elemento Service. Esse possui os elementos: Name, Title, Abstract e OnlineResource. Os elementos Name, Title e Abstract identificam o servidor WXS. O elemento OnlineResource corresponde ao endereço IP do servidor.

O *XML Schema* da Figura 3.6(b) define o elemento Capability. O elemento Capability contém o elemento request seguido dos métodos que o servidor WXS implementa, por sua vez, cada método possui subelementos que descrevem sua forma de acesso.

Por fim, o *XML Schema* da Figura 3.6(c) define o elemento XMLViewsListType. O elemento XMLViewsList contém um elemento operation, o qual define a operação que todas as visões XML publicadas suportam, e um ou mais elementos XMLView, os quais descrevem cada visão que o serviço WXS oferece. O elemento operation deve possuir um subelemento query, indicando que o servidor é capaz de executar consultas sobre visões XML.

Cada elemento XMLView possui os atributos name e schema, os quais são, respectivamente, o nome da visão e o caminho do esquema XML da visão, e um elemento database, o qual descreve os dados de conexão com o banco de dados.

Figura 3.6: (a) *XML Schema* do tipo ServiceType. (b) *XML Schema* do tipo Capability. (c) *XML Schema* do tipo XMLViewsList.

3.2.2 Método getViewType

Dado o nome da visão, o método getViewType retorna seu esquema XML. O esquema da visão XML mostra como consultas podem ser definidas e como as instâncias da visão são geradas.

A Figura 3.7 mostra o *XML Schema* de uma requisição getViewType, a qual contém um elemento getViewType encapsulando zero ou mais elementos ViewName, que corresponde(m) ao(s) nome(s) da(s) visão(ões) a ser(em) descrita(s), e dois atributos: service e outputFormat. O atributo service informa a descrição do serviço. O atributo outputFormat determina o formato do documento de saída. O valor padrão para esse atributo é text/xml; subtype=xml/1.1, indicando que um documento XML válido será gerado.

Se o conteúdo do elemento getViewType é vazio, todas as visões XML publicadas são apresentadas.

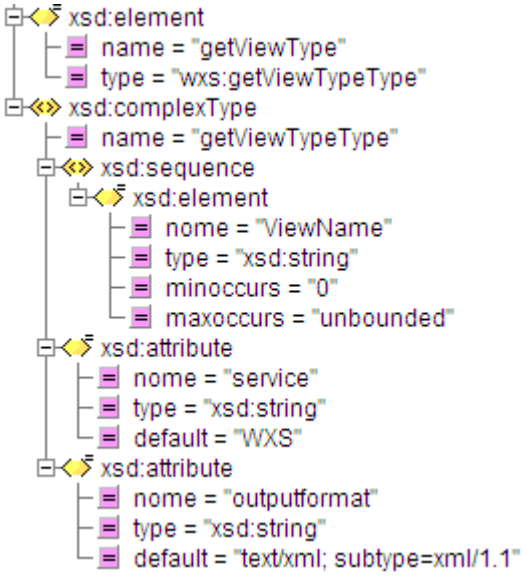


Figura 3.7: *XML Schema* da requisição getViewType.

Considere a visão XML Departamentos publicada em um servidor WXS, cujo esquema é apresentado na Figura 3.8. A visão Departamentos é um conjunto de elementos <Departamento>, os quais são instâncias do tipo TDepts.

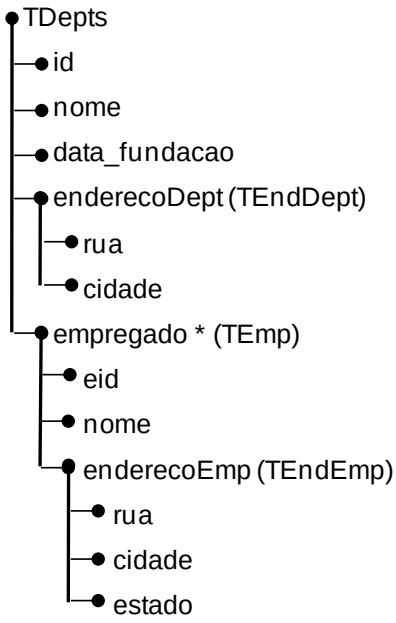


Figura 3.8: Esquema da Visão XML Departamentos.

As Figuras 3.9(a) e (b) mostram um exemplo de requisições getViewType usando o método HTTP POST (formato XML) e HTTP GET, respectivamente, cuja visão XML a ser descrita é Departamentos. Para a tecnologia SOAP, coloca-se a requisição getViewType que está no formato XML no conteúdo do elemento <Body> da mensagem.

A resposta a uma requisição getViewType é um documento *XML Schema* bem-formado e válido, da visão XML correspondente. Por exemplo, o *XML Schema* da visão Departamentos é apresentado na Figura 3.10.

```
<?xml version="1.0"?>
<getViewType
        service="WXS"
        outputformat=" text/xml; subtype=xml/1.1"
        <!-- definição das namespaces --> >
    <ViewName> Departamentos </ViewName>
</getViewType>
```

(a)

```
http://www.ip_do_servidor_wxs:porta/wxs
SERVICE=WXS&
REQUEST=getViewType&
OUTPUTFORMAT=" text/xml; subtype=xml/1.1"&
VIEWNAME= Departamentos
```

(b)

Figura 3.9: (a) Exemplo de uma requisição getViewType utilizando HTTP POST. (b) Exemplo de uma requisição getViewType utilizando o método HTTP GET.

```
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema" elementFormDefault = "qualified"
attributeFormDefault = "unqualified">
    <xsd:element name = "Departamentos" type = "TDepts"/>
    <xsd:complexType name = "TDepts">
        <xsd:sequence>
            <xsd:element nome = "id"/>
            <xsd:element nome = "nome"/>
            <xsd:element nome = "data_fundacao"/>
            <xsd:element nome = "enderecoDept" type = "TEndDept"/>
            <xsd:element nome = "empregado" type = "TEmp" maxoccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = "TEndDept">
        <xsd:sequence>
            <xsd:element nome = "rua"/>
            <xsd:element nome = "cidade"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = "TEmp">
        <xsd:sequence>
            <xsd:element nome = "eid"/>
            <xsd:element nome = "nome"/>
            <xsd:element nome = "enderecoEmp" type = "TEndEmp"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = "TEndEmp">
        <xsd:sequence>
            <xsd:element nome = "rua"/>
            <xsd:element nome = "cidade"/>
            <xsd:element nome = "estado"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figura 3.10: *XML Schema* da visão Departamentos.

3.2.3 Método query

O método query permite consultar uma visão XML usando a linguagem XQuery. Uma consulta XQuery é especificada sobre o esquema de uma visão XML publicada no servidor WXS e o resultado é um documento XML com instâncias do tipo da visão.

A Figura 3.11 mostra o *XML Schema* de uma requisição query, a qual contém um elemento query, cujo valor é uma consulta XQuery, e o atributo outputFormat, cujo valor é text/xml; subtype=xml/1.1.

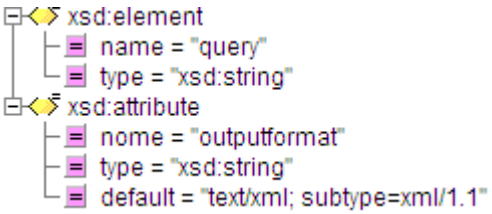


Figura 3.11: *XML Schema* da requisição query.

As Figuras 3.12(a) e (b) mostram um exemplo de requisições query usando o método HTTP POST (formato XML) e HTTP GET, respectivamente. O conteúdo do elemento query é uma consulta XQuery que recupera os NOME dos departamentos que tem CODIGO maior que 1300. A consulta XQuery é dada sobre a visão XML Departamentos apresentada na Figura 3.8.

Para a tecnologia SOAP, coloca-se a requisição query que está no formato XML no conteúdo do elemento <Body> da mensagem.

```
<?xml version="1.0" ?>
<query outputformat=“ text/xml; subtype=xml/1.1”>
      for $i in Departamentos
      where $i/CODIGO > 1300
      return    <Departamento>{
                     $i/Nome
                }</Departamento>
</query>
```

(a)

```
http://www.ip_do_servidor_wxs:porta/wxs
SERVICE=WXS&
OUTPUTFORMAT=“text/xml; subtype=xml/1.1”&
REQUEST=QUERY&
for $i in Departamentos
where $i/CODIGO > 1300
return    <Departamento>{
                $i/Nome
           }</Departamento>
```

(b)

Figura 3.12: (a) Exemplo de uma requisição query utilizando HTTP POST. (b) Exemplo de uma requisição query utilizando o método HTTP GET.

A resposta a uma requisição query é um documento XML bem-formado e válido, o qual é formado por uma coleção de elementos do tipo da visão. Considere, por exemplo, a requisição query apresentada na Figura 3.12. Como mostra a Figura 3.13, a resposta a essa requisição é um documento XML com uma coleção de elementos nome do tipo TDepts da visão.

```
<Departamentos>
   <Departamento>
      <nome> Recursos Humanos </nome>
   </Departamento>
   <Departamento>
      <nome> Jurídico </nome>
   </Departamento>
    ....
</Departamentos>
```

Figura 3.13: Documento XML resultante da requisição query da Figura 3.12.

Capítulo 4
Publicação de visões XML no *RelP*

Neste Capítulo, apresentamos o processo de publicação de visões XML com a ferramenta XML View Publisher (XVP). O Capítulo está organizado da seguinte forma. A Seção 4.1 apresenta nosso formalismo de visões XML e discute como especificar visões XML usando Assertivas de Correspondência. A Seção 4.2 apresenta o algoritmo que gera automaticamente Templates de Consulta a partir Assertivas de Correspondência. A Seção 4.3 apresenta a ferramenta XVP para a publicação de visões XML sobre base relacional. Finalmente, a Seção 4.4 apresenta a conclusão.

4.1 Usando Assertivas de Correspondência para especificar visões XML

Neste trabalho, uma visão XML é especificada por um esquema XML e um conjunto de Assertivas de Correspondência (ACs), as quais especificam o mapeamento entre o esquema da visão e o esquema do banco relacional.

As visões XML devem estar de acordo com um esquema XML pré-definido (*schema-directed publishing*). Deste modo, as Assertivas de Correspondência induzem mapeamentos definidos pela classe de consultas SQL/XML da forma *projeção-seleção-equijunção* (PSE). Essa classe de consultas oferece suporte à maioria dos tipos de reestruturação de dados que são comuns em aplicações de troca de dados.

Seja S um esquema relacional. Uma visão XML, ou simplesmente, uma visão sobre S é uma quadrupla $\mathbf{V} = \langle e, T, R, \mathcal{A} \rangle$, onde:

(i) e é o nome do elemento primário da visão;

(ii) T é o tipo XML do elemento e;

(iii) R é o esquema de uma relação ou o esquema de uma visão relacional de S;

(iv) $\mathcal{A}$ é um conjunto de Assertivas de Correspondência que especifica completamente T em termos de *R*.

O par <e, T> é denominado o esquema da visão **V** e *R* é o esquema da relação pivô da visão, tal que existe mapeamento 1-1 entre as tuplas da tabela/visão pivô e os elementos da visão. Caso não haja relação ou visão que satisfaça a restrição do mapeamento 1-1, devemos primeiro definir uma visão relacional que satisfaça essa restrição.

Em síntese, o processo de geração das ACs é *top down* e recursivo: Primeiro, (i) são definidas as ACs dos elementos do tipo da visão XML com atributos/caminhos da tabela pivô. Em seguinda, (ii) para cada tipo complexo da visão deve-se então recursivamente definir as ACs dos seus elementos com o esquema relacional.

Seja S um esquema relacional, **V**=< e, T, *R*, $\mathcal{A}$ > uma visão sobre S e $\mathcal{A}$ define o mapeamento funcional, denotado por $\mathcal{M}[\mathcal{A}]$, de tuplas de *R* para instâncias de T. A partir da especificação de **V**, pode-se definir um mapeamento funcional, chamado por **Def**[**V**], que mapeia instancias do esquema fonte S para instancias do esquema da visão. Seja σ o estado corrente e $\sigma(R)$ o estado da relação *R*. O valor de **V** sobre σ, **Def**[**V**](σ), é dado por:

Def[**V**](σ) = { **$t** | **$t** é um elemento <e> do tipo T e $\exists$**r** $\in$ $\sigma(R)$ tal que $\mathcal{M}$[A](**r**) $\equiv$ **$t** }

Uma consulta SQL/XML que implementa **Def**[**V**] é dado por:

SELECT XMLELEMENT(“e”, $\tau[R \rightarrow T](r)$) FROM *R r*.

onde, $\tau[R \rightarrow T][r]$ é uma sequência de subconsultas SQL/XML, uma para cada elemento/atributo de T, o qual implementa $\mathcal{M}[\mathcal{A}]$. Dado uma tupla **r** de $\sigma(R)$, $\sigma(\tau[R \rightarrow T][r])(\mathbf{r})$ denota o resultado de uma subconsulta SQL/XML na instância σ, com *r* substituído por **r**. Então, $\sigma(\tau[R \rightarrow T][r])(\mathbf{r}) = \sigma(\mathcal{M}[\mathcal{A}])(\mathbf{r})$.

O formalismo das Assertivas de Correspondência é independente de tecnologia, de modo que outras definições (XQuery [40], XSLT [42], etc...) podem ser facilmente obtidas a partir do mapeamento funcional **Def**[**V**].

Exemplo 4.1

Considere, por exemplo, o esquema relacional Pedidos_BD descrito na Figura 2.1. Suponha a visão XML Pedidos_XML, cujo esquema é apresentado na Figura 2.3, a qual é um conjunto de elementos <pedido> instâncias do tipo TPedido_XML, gerada a partir das tuplas do esquema relacional Pedidos_BD. As Assertivas de Correspondência de Pedidos_XML, as quais especificam TPedido_XML em termos de Pedidos_BD, são apresentadas na Figura 4.1 e são geradas nos passos a seguir:

Passo 1: Os elementos e atributos de **TPedido_XML** são relacionados com os atributos/caminhos de **PEDIDOS_REL**. Por exemplo, a assertiva φ_2:[**TPedido_XML**/DataPedido ≡ **PEDIDOS_REL**/PDATA_PEDIDO] foi gerada relacionando o elemento DataPedido de **TPedido_XML** da visão com o atributo PDATA_PEDIDO da relação **PEDIDOS_REL**. Para o elemento Cliente do tipo complexo, a assertiva φ_3:[**TPedido_XML**/Cliente ≡ **PEDIDOS_REL**/FK_1] foi gerada relacionando o elemento Cliente de **TPedido_XML** da visão com a chave estrangeira FK_1, a qual corresponde ao relacionamento entre as tabelas **PEDIDOS_REL** e **CLIENTES_REL** do esquema do banco. Para o elemento Item do tipo complexo, a assertiva φ_{11}:[**TPedido_XML**/Item ≡ **PEDIDOS_REL**/FK_2^{-1}] foi gerada relacionando o elemento Item de **TPedido_XML** da visão com a chave estrangeira inversa FK_2^{-1}, a qual corresponde ao relacionamento entre as tabelas **PEDIDOS_REL** e **ITENS_REL** do esquema do banco.

Passo 2: Similarmente ao Passo 1, para cada tipo complexo do tipo **TPedido_XML** são geradas as Assertivas. Por exemplo, os elementos e atributos de **TCliente_XML** são relacionados com os atributos/caminhos de **CLIENTES_REL**, os elemento e atributos de **TItem_XML** são relacionados com os atributos/caminhos de **ITENS_REL**, etc. Se um elemento do tipo complexo da visão não possuir correspondência com atributos/caminhos de uma relação, a assertiva assume valor nulo para o atributo/caminho da relação. Por exemplo, no elemento do tipo complexo Endereco de **TCliente_XML**, sua assertiva (φ_5:[**TCliente_XML**/Endereco ≡ **CLIENTES_REL**/**NULL**]) não possui atributo/ caminho correspondente com o esquema do banco.

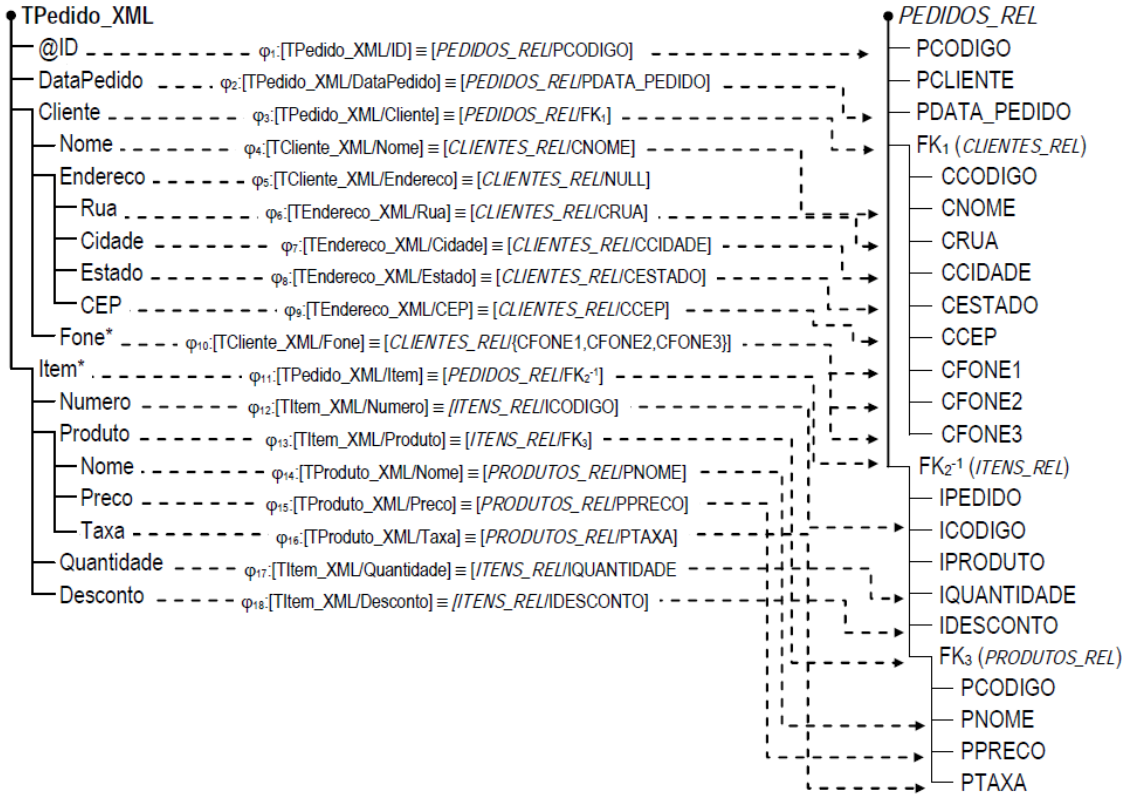


Figura 4.1: Assertivas de Correspondência da visão Pedidos_XML.

4.2 Algoritmo para Geração dos Templates de Consulta

O processo de publicação de uma visão XML no ***RelP*** tem como objetivo gerar os *Templates* de Consulta (TCs) da visão. Os TCs são essenciais para o processo de tradução de consultas, pois eles simplificam e garantem a eficiência na tradução de consulta, uma vez que todo o processamento dos mapeamentos complexos da visão é realizado em tempo de projeto. Assim, a tradução de uma consulta se resume a uma composição dos fragmentos relevantes para construir a consulta SQL/XML correta.

Os TCs são fragmentos de consultas SQL/XML gerados a partir das Assertivas de Correspondência da visão. Os TCs são agrupados em *tabelas template*, sendo uma para cada tipo complexo da visão. O algoritmo que automaticamente gera os TCs, chamado **GeraTC**, é apresentado na Listagem 4.1.

Primeiramente, o algoritmo **GeraTC**, constrói a *tabela template* do tipo da visão. Desse modo, para cada caminho a partir do tipo da visão, é gerada uma subconsulta SQL/XML que constrói uma instância do caminho a partir dos dados

relacionais (linhas 5 – 10). Caso o caminho seja um elemento do tipo da visão, a subconsulta SQL/XML é gerada com base na AC desse elemento com o banco. Caso contrário, a consulta SQL/XML é gerada com base na composição das ACs dos elementos do caminho. Em seguida, o algoritmo gera de forma similar as *tabelas template* para os outros tipos complexos da visão (linhas 13 – 22).

Em casos em que os elementos da visão são complexos, é necessário gerar uma subconsulta SQL/XML que constrói o elemento XML sem conteúdo e outra com conteúdo completo, ou seja, com todos seus subelementos. O algoritmo **GeraTC** utiliza a subconsulta SQL/XML sem conteúdo quando o elemento complexo é reestruturado na consulta, pois seu conteúdo será gerado em seguida com base nos subelementos selecionados na consulta XQuery.

O algoritmo **GeraSubSQL/XML**, apresentado na Listagem 4.2, reescreve um caminho de um elemento XML da visão em uma subconsulta SQL/XML correspondente, como explicado anteriormente. O **GeraSubSQL/XML** utiliza como base as ACs para encontrar o(s) atributo(s)/caminho(s) relacional correspondente ao elemento XML.

Entrada: V=<e, T_e, *R*, $\mathcal{A}$>.
Saída: Templates de Consulta da visão.

```
1.   Seja: Q um conjunto de templates de uma tabela template e r uma alias para a tabela R;
2.
3.   //Geração da tabela template para o esquema da visão.
4.   Q := { T_T }, onde T_T é a tabela template para o tipo T_e;
5.   Para cada caminho δ de T_e faça
6.       Se T_e é complexo Então
7.           T_T [ δ ] := GeraSubSQL/XML( e, A, Falso);
8.       Fim Se;
9.       T_T [ δ ] := GeraSubSQL/XML( e, A, Verdadeiro);
10.  Fim Para;
11.
12.  //Geração das tabelas template para os outros tipos complexos.
13.  Para cada tipo complexo T'_e a partir de T_e faça
14.      Q := Q ∪ { T_T' }, onde T_T' é a tabela template para o tipo T'_e;
15.      Para cada caminho δ de T'_e faça
16.          Se (T_e é complexo) Então
17.              T_T'[ δ ] := GeraSubSQL/XML( e, A, Falso);
19.          Fim Se;
20.      T_T' [ δ ] := GeraSubSQL/XML(e, A, Verdadeiro);
21.      Fim Para;
22.  Fim Para;
23.  Retorne Q;
```

Listagem 4.1: Algoritmo **GeraTC**.

Entrada: Elemento XML e, assertiva de correspondência $\mathcal{A}$ do caminho δ e um booleano CONTENT que indica se o elemento será criado com seu conteúdo ou não.
Saída: Subconsulta SQL/XML **C**.

Se (CONTENT) **então**
  Caso 1: **Se** e simples ocorrência, T_e é do tipo atômico e $\mathcal{A}$ = **a**, **então**
    C = "XMLFOREST(**r.a** AS \" e \")";
  Caso 2: **Se** e é simples ocorrência, T_e é do tipo atômico e $\mathcal{A}$ = φ.**a**, **então**
    C = "SELECT XMLELEMENT((\"e \", $\mathbf{r}_n$.**a** FROM **Joinφ**(**r**)))";
  Caso 3: **Se** e é simples ocorrência, T_e é do tipo geométrico e $\mathcal{A}$ = **a**, **então**
    C = "XMLFOREST(***g*** (**r .a**) AS \" e \")";
  Caso 4: **Se** e é múltipla ocorrência, T_e é do tipo atômico e $\mathcal{A}$ = {$\mathbf{a}_1$,...,$\mathbf{a}_n$}, **então**
    C = "XMLCONCAT(XMLFOREST(**r.a**$_1$ AS \" e \")," + …+ "XMLFOREST(\"e \", **r.a**$_n$ AS \" e \"))";
  Caso 5: **Se** e é múltipla ocorrência, T_e é tipo atômico e $\mathcal{A}$ = φ/ {$\mathbf{a}_1$,...,$\mathbf{a}_n$}, **então**
    C = "SELECT XMLCONCAT((SELECT XMLFOREST($\mathbf{r}_n$.$\mathbf{a}_1$ AS \" e \", … , $\mathbf{r}_n$.$\mathbf{a}_n$ AS \" e \")
      FROM **Joinφ**(**r**)))";
  Caso 6: **Se** e é múltipla ocorrência, T_e é tipo atômico e $\mathcal{A}$ = φ/ **a**, **então**
    C = "(SELECT XMLAGG(XMLFOREST($\mathbf{r}_n$.**a** AS \" e \") FROM **Joinφ**(**r**))";
  Caso 7: **Se** e é simples ocorrência, T_e é tipo complexo e $\mathcal{A}$ = φ, **então**
    C = **C** + "(SELECT XMLELEMENT(\"e \", " +
      + **Para cada** subelemento e' de T_e **faça**
        GeraSubSQL/XML (e', $\mathcal{A}$, Verdadeiro) + ") FROM **Joinφ**(**r**)))"
  Caso 8: **Se** e é múltipla ocorrência, T_e é tipo complexo e $\mathcal{A}$ = φ, **então**
    C = **C** + "(SELECT XMLAGG(XMLELEMENT(AS \" e \", " +
      + **Para cada** subelemento e' de T_e **faça**
        GeraSubSQL/XML (e', $\mathcal{A}$, Verdadeiro) +")) FROM **Joinφ**(**r**))"
  Caso 9: **Se** e é simples ocorrência, T_e é tipo complexo e $\mathcal{A}$ = NULL, **então**
    C = **C** + "XMLELEMENT(\"e \", " +
      + **Para cada** subelemento e' de T_e **faça**
        GeraSubSQL/XML (e', $\mathcal{A}$, Verdadeiro) + ")"
Senão
  Caso 1: **Se** e é simples ocorrência, T_e é tipo complexo e $\mathcal{A}$ = φ, **então**
    C = "(SELECT XMLELEMENT(\"e \", %content%) FROM **Joinφ**(**r**))"
  Caso 2: **Se** e é múltipla ocorrência, T_e é tipo complexo e $\mathcal{A}$ = φ, **então**
    C = "(SELECT XMLAGG(XMLELEMENT(\" e \", %content%")) FROM **Joinφ**(**r**))"
  Caso 3: **Se** e é simples ocorrência, T_e é tipo complexo e $\mathcal{A}$ = NULL, **então**
    C = "XMLELEMENT(\"e \", %content%)";
Fim Se

Retorne C;

Nota:

(i) Se o booleano CONTENT for falso a subconsulta SQL/XML é criada sem seu conteúdo.

(ii) φ é um caminho referencial de R para R', na forma φ_1. ... . φ_{n-1}.

(iii) **Joinφ**(**r**) é definida pelos seguintes fragmentos SQL:

R_1 $\mathbf{r}_1$,..., R_n $\mathbf{r}_n$ WHERE $\mathbf{r}.a_k^{[1]} = \mathbf{r}_1.b_k^{[1]}$, $1 \le k \le m_1$, AND $\mathbf{r}_{i-1}.a_k^{[i]} = \mathbf{r}_i.b_k^{[i]}$, $1 \le k \le m_i$, $2 \le i \le n$.

Tal que, dado uma tupla **r** de *R*, então $\mathbf{r}.\varphi$ = SELECT $\mathbf{r}_n$ FROM **Join**φ(**r**).

(iv) As funções da linguagem SQL/XML [22, 23]:
- XMLElement() constrói elementos XML;
- XMLForest() constrói uma sequencia de elementos XML, sem valores nulos;
- XMLConcat() concatena elementos XML; e
- XMLAgg() agrega elementos XML.

(v) ***g*** (**r.a**) é uma função (proprietária do SGBD) que recebe um atributo do tipo geométrico e gera o fragmento GML correspondente. O Oracle, por exemplo, provê a função SDO_UTIL.TO_GMLGEOMETRY(), a qual gera elementos GML a partir do tipo geométrico proprietário SDO_GEOMETRY.

Listagem 4.2: Algoritmo **GeraSubSQL/XML**.

Exemplo 4.2

Considere por exemplo, o esquema da visão Pedidos_BD, apresentado na Figura 2.3, e as Assertivas de Correspondência da Figura 4.1. Para a geração dos Templates de Consulta com o algoritmo **GeraTC**, considere os passos a seguir:

Passo 1: Geração da tabela template do tipo da visão. O algoritmo gera a *tabela template* TPedido_XML e verifica cada caminho da visão a partir de TPedido_XML gerando a subconsulta SQL/XML correspondente. A Tabela 4.1 mostra as iterações do algoritmo **GeraTC** (linhas 5 – 10) gerando as respectivas subconsultas SQL/XML para os caminhos do tipo TPedido_XML. Na Tabela, a coluna Processamento apresenta os parâmetros de entrada e a parte do algoritmo **GeraSubSQL/XML** que é executada.

Iteração	Caminho do tipo XML	Processamento	Saída
#1	@ID	Assertiva: TPedido_XML/@ID ≡ Pedidos_rel/pcodigo Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(P.PCODIGO AS "ID")*
#2	DataPedido	Assertiva: TPedido_XML/DataPedido ≡ Pedidos_rel/pdata_pedido Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(P.PDATA_PEDIDO AS "DataPedido")*
#3	Cliente*	Assertiva: TPedido_XML/Cliente ≡ Pedidos_rel/FK_1 Content: Falso GeraSubSQL/XML: Caso 1 (linha 28).	*(SELECT XMLELEMENT("Cliente", %content%) FROM CLIENTES_REL WHERE C.CCODIGO = P.PCLIENTE)*

#4	Cliente	Assertiva: TPedido_XML/Cliente ≡ Pedidos_rel/FK_1 Content: Falso GeraSubSQL/XML: Caso 7 (linha 15). **Chamada recursiva na ordem dos elementos:** 1. Content = Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2). 2. Content = Verdadeiro GeraSubSQL/XML: Caso 9 (linha 23). 3. Content = Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2). GeraSubSQL/XML: Caso 1 (linha 2). GeraSubSQL/XML: Caso 1 (linha 2). GeraSubSQL/XML: Caso 1 (linha 2). GeraSubSQL/XML: Caso 4 (linha 8).	(SELECT XMLELEMENT("Cliente", XMLFOREST(C.CNOME AS"Nome"), XMLELEMENT("Endereco", XMLFOREST(C.RUA AS "Rua"), XMLFOREST(C.CIDADE AS Cidade"), XMLFOREST(C.ESTADO AS "Estado"), XMLFOREST(C.CEP AS "CEP")), XMLFOREST(C.FONE1 AS "Fone", C.FONE2 AS "Fone", C.FONE3 AS "Fone")) *FROM CLIENTES_REL C* *WHERE C.CCODIGO = P.PCLIENTE)*
#5	Cliente/Nome	Assertiva: TCliente_XML/Nome ≡ Clientes_rel/cnome Content: Verdadeiro GeraSubSQL/XML: no Caso 2 (linha 4).	(SELECT XMLELEMENT("Nome", C.CNOME) FROM CLIENTES_REL C *WHERE C.CCODIGO =P.PCLIENTE)*
#6	Cliente/Endereco*	Assertiva: TCliente_XML/Nome ≡ Clientes_rel/NULL Content: Falso GeraSubSQL/XML: no Caso 1 (linha 28).	(SELECT XMLELEMENT("Endereco", *%content%*) *FROM CLIENTES_REL C* *WHERE C.CCODIGO = P.PCLIENTE)*
...	...	...	...
#12	Cliente/Fone	Assertiva: TCliente_XML/Fone ≡ Clientes_rel/{CFONE1, CFONE2, CFONE3} Content: Verdadeiro GeraSubSQL/XML: Caso 5 (linha 10).	(SELECT XMLCONCAT(XMLFOREST (C.CFONE1 AS"Fone", XMLFOREST (C.CFONE2 AS"Fone", XMLFOREST (C.CFONE3 AS"Fone",)) FROM CLIENTES_REL C *WHERE C.CCODIGO =P.PCLIENTE)*
#13	Item*	Assertiva: TPedido_XML/Item ≡ Pedidos_rel/FK_2^{-1} CONTENT = Falso GeraSubSQL/XML: Caso 2 (linha 30).	(SELECT XMLAGG(XMLELEMENT("Item", %content%) *FROM ITENS_REL I WHERE* *I.IPEDIDO = P.CCODIGO)*
...	...	...	...
#15	Item/Produto*	Assertiva: TItem_XML/Produto ≡ Produtos_rel/FK_3 CONTENT = Falso GeraSubSQL/XML: Caso 2 (linha 30).	(SELECT XMLAGG(XMLELEMENT("Produto", %content%)) FROM ITENS_REL I, PRODUTOS_REL PR *WHERE I.IPEDIDO = P.CCODIGO* *AND PR.PCODIGO = I.IPRODUTO)*
...	...	...	...

Tabela 4.1: Iterações de **GeraTC** (linhas 5 – 10) para geração da *tabela template* **TPedido_XML**.

Passo 2: Geração das outras *tabelas template*. O algoritmo gera as *tabelas template* **TCliente_XML**, **TEndereco_XML, TItem_XML** e **TProduto_XML**. Para cada *tabela template* criada, o algoritmo verifica cada caminho a partir do seu tipo e gera as subconsultas SQL/XML correspondentes. As Tabelas 4.2, 4.3, 4.4 e 4.5 mostram algumas iterações do algoritmo **GeraTC** (linhas 13 – 22) gerando subconsultas SQL/XML nas *tabelas template* **TCliente_XML**, **TEndereco_XML, TItem_XML** e **TProduto_XML**, respectivamente.

Iteração	Caminho do tipo	Processamento	Saída
#1	Nome	Assertiva: TCliente_XML/Nome ≡ Clientes_rel/cnome Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CNOME AS "Nome")*
#2	Endereco*	Assertiva: TCliente_XML/Endereco ≡ Clientes_rel/NULL CONTENT: Falso GeraSubSQL/XML: Caso 3 (linha 32).	*XMLELEMENT("Endereco",%content%)*
#3	Endereco/Rua	Assertiva: TEndereco_XML/Rua ≡ Clientes_rel/crua Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CRUA AS "Rua")*
#4	Endereco/Cidade	Assertiva: TEndereco_XML/Cidade ≡ Clientes_rel/ccidade Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CCIDADE AS "Cidade")*
...	...	...	...

Tabela 4.2: Iterações de **GeraTC** (linhas 13 – 22) para geração da *tabela template* **TCliente_XML**.

Iteração	Caminho do tipo	Processamento	Saída
#1	Rua	Assertiva: TEndereco_XML/Rua ≡ Clientes_rel/crua Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CRUA AS "Rua")*
#2	Cidade	Assertiva: TEndereco_XML/Cidade ≡ Clientes_rel/ccidade Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CCIDADE AS "Cidade")*
#3	Estado	Assertiva: TEndereco_XML/Estado ≡ Clientes_rel/cestado Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CESTADO AS "Estado")*
#4	CEP	Assertiva: TEndereco_XML/Cep ≡ Clientes_rel/ccep Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(C.CCEP AS "Cep")*

Tabela 4.3: Iterações de **GeraTC** (linhas 13 – 22) para geração da *tabela template* **TEndereco_XML**.

Iteração	Caminho do tipo	Processamento	Saída
#1	Numero	Assertiva: TItem_XML/Numero ≡ Itens_rel/icodigo Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(I.ICODIGO AS "Numero")*
#2	Produto*	Assertiva: TItem_XML/Produto ≡ Itens_rel/FK_3 Content: Falso GeraSubSQL/XML: Caso 1 (linha 28).	*(SELECT XMLELEMENT("Produto", %content%) FROM PRODUTOS_REL PR WHERE PR.PCODIGO = I.IPRODUTO)*
...	...	...	...
#7	Quantidade	Assertiva: TItem_XML/Quantidade ≡ Itens_rel/iquantidade Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(I.IQUANTIDADE AS "Quantidade")*
#8	Desconto	Assertiva: TItem_XML/Desconto ≡	*XMLFOREST(I.IDESCONTO AS "Desconto")*

		Itens_rel/idesconto Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	

Tabela 4.4: Iterações de **GeraTC** (linhas 13 – 22) para geração da *tabela template* **TItem_XML**.

Iteração	Caminho do tipo	Processamento	Saída
#1	Nome	Assertiva: TProduto_XML/Nome ≡ Produtos_rel/pnome Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(PR.PRUA AS "Rua")*
#2	Preco	Assertiva: TProduto_XML/Preco ≡ Produtos_rel/cpreco Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(PR.PPRECO AS "Preco")*
#3	Taxa	Assertiva: TProduto_XML/Taxa ≡ Produtos_rel/ctaxa Content: Verdadeiro GeraSubSQL/XML: Caso 1 (linha 2).	*XMLFOREST(PR.PTAXA AS "Taxa")*

Tabela 4.5: Iterações de **GeraTC** (linhas 13 – 22) para geração da *tabela template* **TProduto_XML**.

4.3 Ferramenta *XML View Publisher*

Neste trabalho, propomos a ferramenta *XML View Publisher* (XVP), para a publicação de visões XML no ***RelP***. Com a XVP, através de sua interface gráfica, os usuários podem criar e publicar uma visão XML no ***RelP*** de forma intuitiva, sem a necessidade de ter o conhecimento profundo do formalismo das ACs.

Uma visão geral da arquitetura da XVP pode ser vista na Figura 4.2. Na arquitetura, o componente *Gerador de Mapeamento* gera o documento de mapeamentos, o qual contém o conjunto das Assertivas de Correspondência que especifica completamente o esquema da visão XML em termos do esquema relacional. As ACs são armazenadas em um documento XML, AC.xml, cuja estrutura está descrita no anexo A.

Além do componente *Gerador de Mapeamento*, a XVP apresenta o componente *Gerador de Templates*, o qual recebe como entrada o documento das ACs e gera os Templates de Consulta da visão. A XVP implementa o algoritmo **GeraTC** discutido na Seção 4.2. Os TCs é armazenado em um documento XML, TC.xml, cuja estrutura está descrita no anexo A.

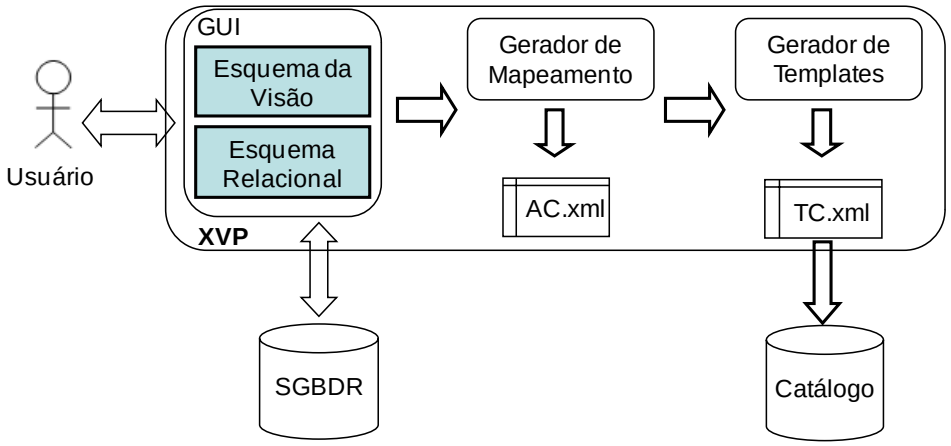


Figura 4.2: Arquitetura da ferramenta *XML View Publisher*.

A ferramenta XVP permite aos usuários duas formas de publicar visões XML no ***RelP***: ***Publicação automática da visão*** e ***Publicação semi-automática da visão***, as quais são discutidas a seguir.

4.3.1 Publicação automática de visão com XML View Publisher

O processo de *publicação automática* de uma visão com XVP, como mostra a Figura 4.3, é realizado nos seguintes passos. Primeiro, (1) o usuário informa os dados de conexão com o banco de dados, (2) escolhe a tabela pivô e (3) seleciona os atributos do esquema relacional que irão compor o esquema da visão XML. Em seguida, (4) baseado nos atributos selecionados, a XVP gera as Assertivas de Correspondência e em seguida os Templates de Consulta da visão. Por fim, (5) o esquema e os TCs da visão são registrados e armazenados através do componente *Catalogue* do ***RelP***.

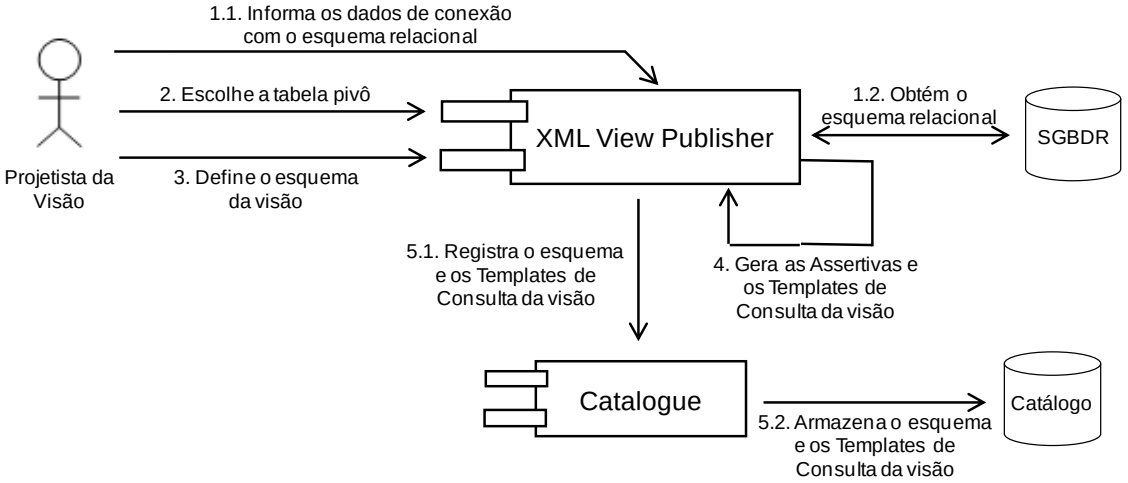


Figura 4.3: Passos para publicação automática de uma visão XML no ***RelP***.

Exemplo 4.3

Considere por exemplo, que o usuário deseje publicar uma visão XML a partir do banco de dados Pedidos_BD apresentado na Figura 2.1. Utilizando a XVP, os seguintes passos são necessários:

Passo 1: Via Interface Gráfica apresentada na Figura 4.4, o usuário se conecta ao banco de dados relacional através da interface JDBC e escolhe a tabela pivô. A Figura 4.4 apresenta em ① os dados de conexão com banco de dados Pedidos_BD, e mostra em ②, a tabela pivô Pedidos_rel selecionada.

Passo 2: Em *Esquema do BD* da XVP, o usuário seleciona os atributos das tabelas, relacionadas pela relação de chave e, através do botão *Construir Esquema da Visão*, é gerado o esquema da visão XML. A Figura 4.4 mostra em ③, o esquema da visão XML.

Passo 3: Em *Publicar visão*, são geradas as ACs, e em seguida, são gerados os TCs. Por fim, XVP registra e armazena o esquema e os TCs da visão através do componente *Catalogue*.

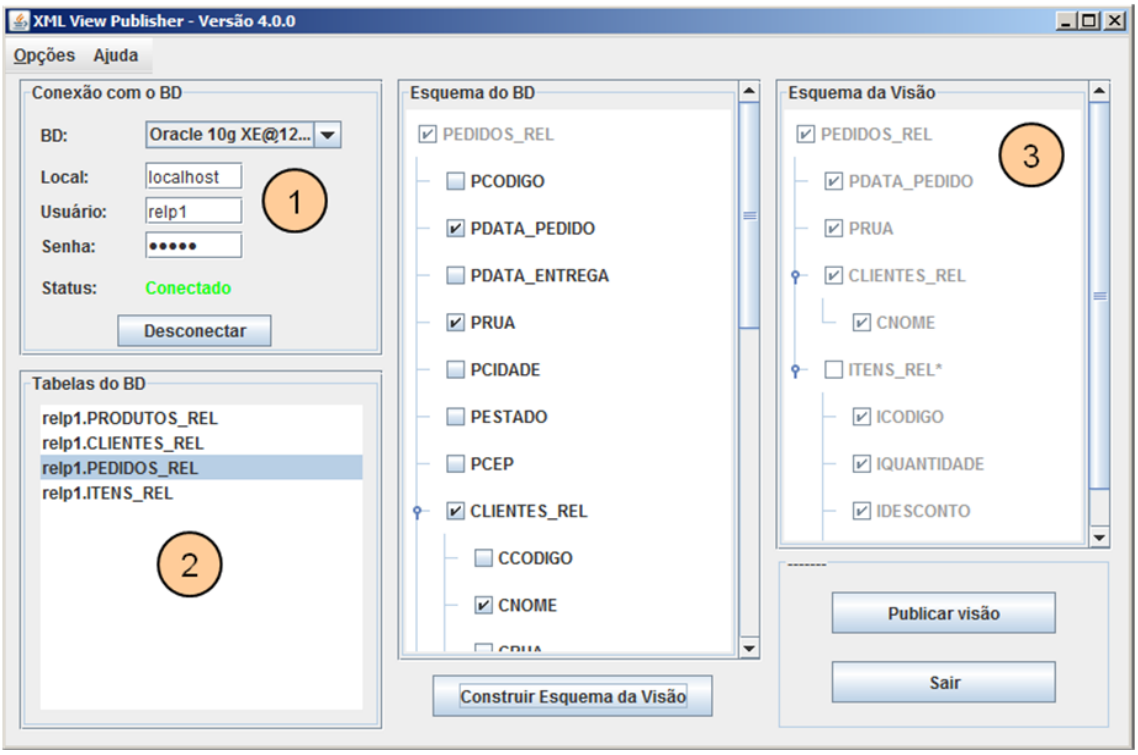


Figura 4.4: Interface da ferramenta *XML View Publisher*.

A publicação automática de uma visão XML permite somente a criação de tipos complexos correspondentes às tabelas relacionais. Por exemplo, na visão Pedidos_XML, apresentada na Figura 2.3, seria impossível criar o tipo complexo TEndereco na XVP, pois TEndereco não é mapeado para nenhuma tabela relacional.

Dessa forma, para criação e publicação de visões XML mais complexas deve ser utilizado o processo de publicação semi-automática de visões XML, discutido a seguir.

4.3.2 Publicação semi-automática de visão com XML View Publisher

O processo de *publicação semi-automática* de uma visão com XVP, como mostra a Figura 4.5, é realizado nos seguintes passos. Primeiro, (1) o usuário define o esquema da visão. Em seguida, (2) o usuário informa os parâmetros de acesso ao banco de dados e o esquema relacional é obtido. Então, (3) o usuário define as ACs entre o esquema da visão e o esquema relacional. Com base nas ACs da visão (4), a XVP gera os TCs da

visão. Por fim, (5) o esquema e os TCs da visão são registrados e armazenados no *Catalogue* do ***RelP***.

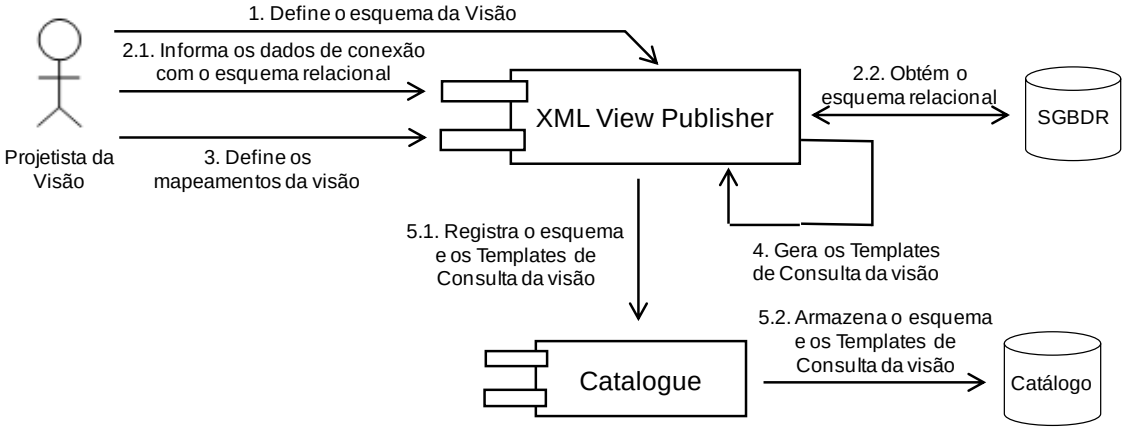


Figura 4.5: Passos para publicação semi-automática de uma visão XML no ***RelP***.

Exemplo 4.4

Considere por exemplo, que o usuário deseje publicar uma visão XML a partir do banco de dados Pedidos_BD apresentado na Figura 2.1. Utilizando a XVP, os seguintes passos são necessários:

Passo 1: O usuário define o tipo XML do elemento primário da visão. XVP oferece uma interface gráfica simples que permite a definição de tipos XML.

Passo 2: O usuário seleciona, de uma lista de tabelas e visões relacionais do esquema do banco, a tabela ou visão pivô, tal que existe um mapeamento 1-1 entre as tuplas de tabela/visão pivô e os elementos da visão.

Passo 3: O usuário especifica as Assertivas de Correspondência que definem o relacionamento entre os elementos e atributos do tipo XML da visão e os atributos/caminhos da tabela pivô. A Figura 4.6 mostra uma tela da interface gráfica da XVP que suporta a definição de Assertivas de Correspondência da visão. O esquema do lado esquerdo da Figura é o esquema XML da visão Pedidos_XML, e do lado direito tem-se o esquema da relação pivô PEDIDOS_REL do esquema relacional PEDIDOS_BD da Figura 2.1.

Passo 4: A XVP gera automaticamente, baseado nas Assertivas de Correspondência, os Templates de Consulta da visão XML, e registra e armazena o esquema e os TCs da visão através do componente *Catalogue*.

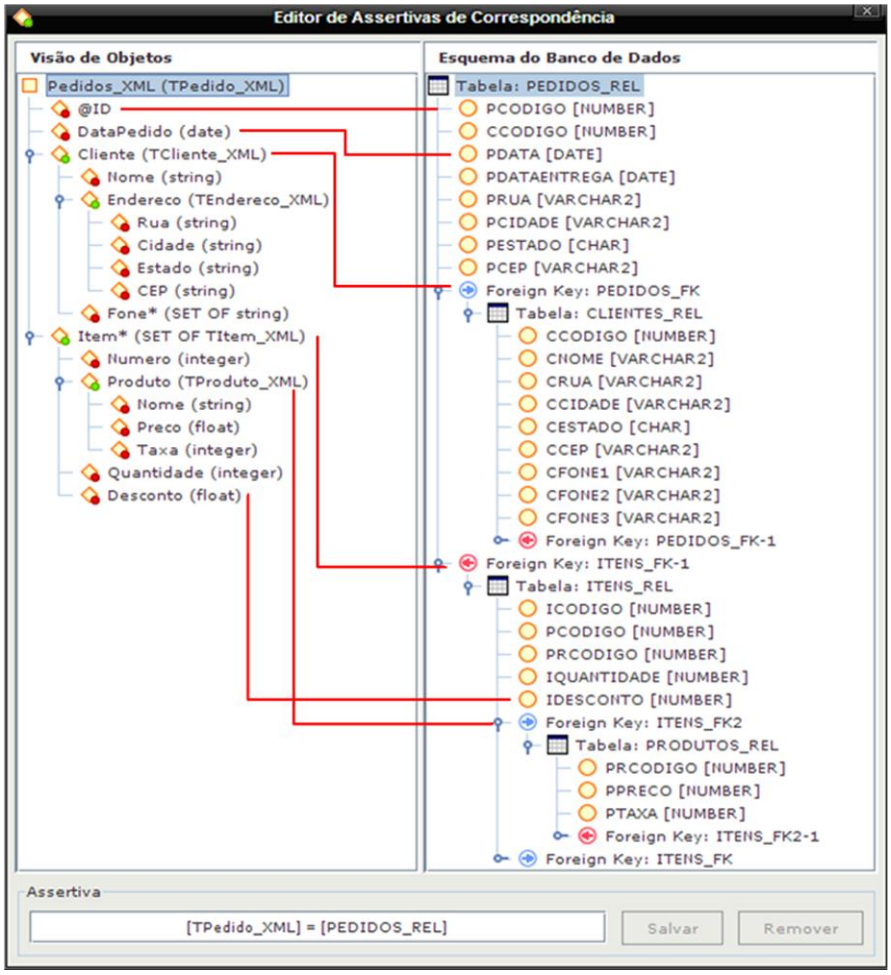


Figura 4.6:Editor das Assertivas de Correspondência da XVP.

4.4 Conclusão

Neste capítulo, definimos formalmente as visões XML publicadas no ***RelP***. Uma visão XML é especificada por um esquema XML e um conjunto de Assertivas de Correspondência (ACs), as quais especificam o mapeamento entre o esquema da visão e o esquema do banco relacional.

Mostramos também, o algoritmo **GeraTC**, o qual gera os Templates de Consulta a partir das Assertivas de Correspondência da visão. Em seguida, mostramos os passos do processo de geração dos TCs para a visão XML Pedidos_XML.

Finalmente, apresentamos a ferramenta *XML View Publisher* (XVP), que permite ao usuário publicar automaticamente e semi-automaticamente visões XML. A XVP facilita a tarefa de publicação de visões XML no ***RelP***, já não é necessário ter o conhecimento profundo do formalismo das ACs.

Capítulo 5
Tradução de Consultas XQuery no *RelP*

*Neste Capítulo, apresentamos o processo de tradução de consultas XQuery no framework **RelP**. O Capítulo está organizado da seguinte forma. A Seção 5.1 descreve o processo geral de tradução de consultas no **RelP**. A Seção 5.2 mostra o algoritmo **GeraTPQX**, o qual gera uma Tree Pattern Query a partir de uma consulta XQuery. A Seção 5.3 mostra o algoritmo **GeraSQL/XML**, o qual gera uma consulta SQL/XML a partir de uma Tree Pattern Query A Seção 5.4 mostra um exemplo de tradução de uma Consulta XQuery em uma Consulta SQL/XML. A Seção 5.5 apresenta a ferramenta XQuery Browser, o qual facilita a construção de consultas XQuery no **RelP**. Finalmente, a Seção 5.6 apresenta a conclusão.*

5.1 Consultando Visões XML no *RelP*

No ***RelP***, consultar uma visão XML consiste em enviar uma consulta XQuery sobre o esquema da visão através do método query do serviço Web *WXService*. O usuário que não domina a linguagem XQuery pode utilizar a ferramenta *XQuery Browser* (vide Seção 5.5) para formular graficamente uma consulta XQuery de forma intuitiva.

O cenário típico de consulta de uma visão XML usando a ferramenta *XQuery Browser* consiste dos passos apresentados na Figura 5.1. Primeiro, (1) o usuário seleciona o esquema de uma visão XML através da ferramenta *XQuery Browser*, e, em seguida, (2) define graficamente a consulta XQuery sobre o esquema da visão. Depois, a ferramenta (3) envia a consulta do método query do serviço Web *WXService*. Logo após, o componente *Query Translator* (4) consulta o componente *Catalogue* para obter os Templates de consulta da visão e (5) traduz a consulta XQuery em uma consulta SQL/XML correspondente, a qual é executada pelo SGBDR. Finalmente, o resultado XML (6) é enviado para o *XQuery Browser* e apresentado ao usuário.

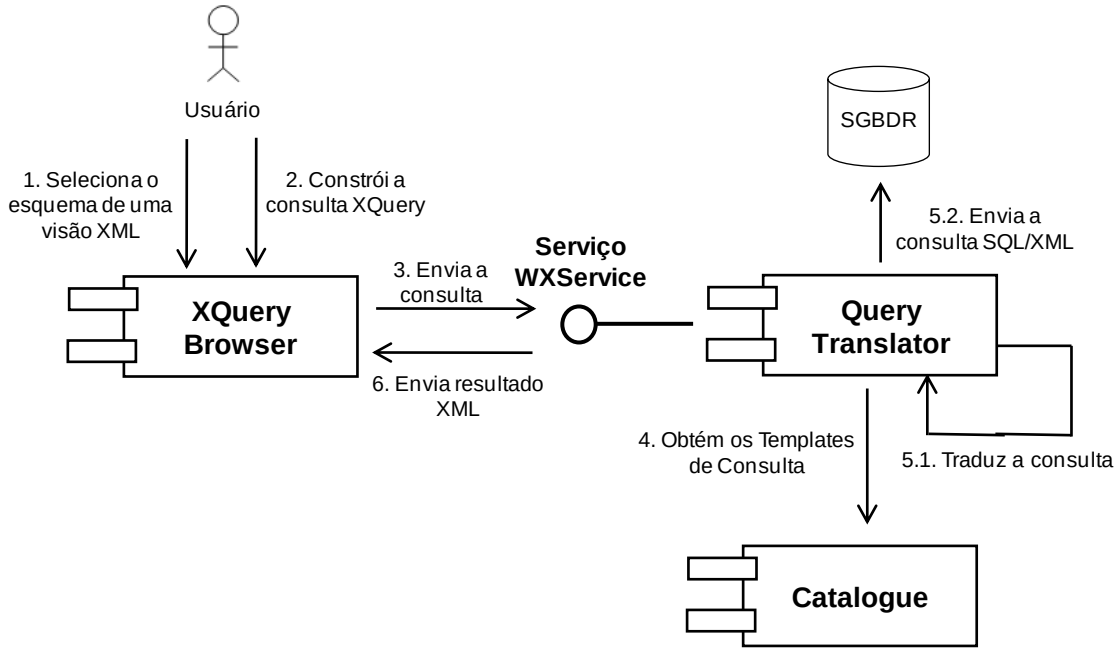


Figura 5.1: Consultando visões XML no ***RelP***.

O ***RelP*** traduz uma consulta XQuery Qx definidas sobre o esquema de uma visão XML em uma consulta SQL/XML Qs sobre o esquema do banco de dados relacional. Qs é uma tradução correta de Qx, o que significa que o resultado da execução de Qx sobre a materialização da visão XML é equivalente ao resultado da execução de Qs sobre o banco de dados, em um dado instante.

A tradução de consulta XQuery em uma consulta SQL/XML é realizada em dois passos. Primeiro, (i) uma *Tree Pattern Query* (TPQ) é gerada a partir da consulta XQuery. Em seguida, (ii) uma consulta SQL/XML é gerada a partir da TPQ. Em nossa abordagem, utilizamos o modelo TPQ para restringir o escopo da consulta XQuery e para facilitar a geração da consulta SQL/XML. Além disso, utilizamos os Templates de Consulta (vide Capítulo 4) para simplificar e garantir a eficiência do processo de tradução. A seguir, detalhamos os passos do processo de tradução de consulta no ***RelP***.

5.2 Algoritmo para Geração de uma *Tree Pattern Query*

O ***RelP*** restringe a expressividade de consultas XQuery ao usuário para garantir eficiência no processamento de consulta. Para tanto, o ***RelP*** aceita somente consultas que possam ser modeladas como *Tree Pattern Queries*. Dessa forma, tais consultas não podem, por exemplo, ser definidas utilizando as cláusulas Let ou Order by, expressões de caminhos com // ou ainda predicados definidos entre colchetes.

Uma TPQ é uma árvore com as seguintes características:

(i) Os nós são rotulados por nomes de elementos/atributos ou valores de dados;

(ii) Os nós rotulados por uma seta são chamados Nós-Resultado. A seta indica que o nó correspondente é parte do resultado da consulta;

(iii) Os elementos multivalorados são rotulados pelo símbolo *.

Por exemplo, a Figura 5.2 apresenta uma árvore TPQ, a qual representa uma consulta XQuery que retorna o id e a data do pedido de todos os pedidos do cliente cujo nome é Smith. Dessa forma, o nó rotulado @ID e DataPedido é definido como um Nó-Resultado e os nós Cliente, nome e "Smith" juntamente com a aresta valorada entre os dois último nós definem o predicado da consulta.

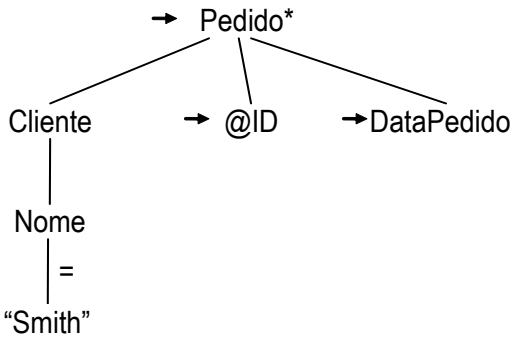


Figura 5.2: Exemplo simples de uma *Tree Pattern Query*.

Para gerar uma TPQ a partir de uma consulta XQuery, implementamos o algoritmo **GeraTPQX**, apresentado na Listagem 5.1. Em resumo, o algoritmo recebe como parâmetro a consulta XQuery e gera a TPQ correspondente em três etapas:

(i) **Análise da cláusula FOR**: o nó raiz da TPQ é obtido a partir do elemento referenciado na cláusula FOR.

(ii) **Análise da cláusula WHERE**: os predicados são analisados e adicionados à TPQ. O algoritmo verifica a expressão de caminho de cada predicado. Para cada elemento do caminho, é adicionado um nó correspondente na TPQ, caso o nó ainda não esteja presente na árvore.

(iii) **Análise da cláusula RETURN**: Para cada elemento da cláusula RETURN, o algoritmo "marca" e/ou adiciona o nó na TPQ correspondente ao elemento. Os nós marcados, chamados de Nós-Resultado, fazem parte do resultado da consulta. Nesta fase, para facilitar a tradução da TPQ na consulta SQL/XML correspondente, os nós que representam elementos multivalorados também são marcados com um *.

Entrada: Consulta XQuery Qx
Saída: Tree Pattern Query

```
1.    Seja: TPQ uma Tree Pattern Query;
2.    Seja P = (δ θ v) um predicado, onde:
3.        δ é uma expressão de caminho na forma e1/.../en, n≥1,
4.        θ é um operador,
5.        v é um valor escalar.
6.
7.    //Análise da cláusula FOR
8.    TPQ.addRaiz( Qx/FOR );
9.
10.   //Análise da cláusula WHERE
11.   Para cada predicado P = (δ θ v) de Qx/WHERE faça
12.       Aux := TPQ.raiz( );
13.           Para cada elemento e de δ = e1/.../en faça
14.               Se (e existe na TPQ) então
15.                   Aux := e;
16.               Senão
17.                   Aux.addFilho(e);
18.                   Aux := e;
19.               Fim Se
20.           FimPara
21.       Aux. addFilhoComArestaValorada( v , θ );
22.   FimPara
23.
24.   //Adiciona as operações entre os predicados.
25.   Para cada operador O entre predicados Pi e Pi+1 de Qx faça
26.       TPQ .addOperador ( O, Pi, Pi+1 );
27.   Fim Para
28.
29.   //Análise da cláusula RETURN
30.   Para cada elemento E de Qx/RETURN faça
31.       Caso 1: Se E existe na TPQ então
32.           TPQ.marcarComNóResultado( E );
33.       Caso 2: Se E é subconsulta XQuery então
34.           TPQ.addFilho (GeraTPQX ( E ));
35.       Default:
36.           TPQ.addFilho( E );
37.           TPQ.marcarComoNóResultado( E );
38.       Fim Caso;
39.
40.       Se (E é o elemento raiz da TPQ) ou (E é um elemento multivalorado) então
41.           TPQ.marcarComoNóMultivalorado( E, verdadeiro );
42.       Fim Se
43.   Fim Para
44.
45.  Retorne TPQ;
```

Nota:
(i) **addFilhoComArestaValorada** é uma função que adiciona um nó filho v com aresta valorada θ.

Listagem 5.1: Algoritmo **GeraTPQX**.

5.3 Algoritmo para Geração de uma Consulta SQL/XML

No segundo passo do processo de tradução de consultas, o componente *Query Translator* percorre a TPQ buscando os Nós-Resultado, e convertendo-os, através dos Templates de Consulta, em subconsultas SQL/XML. Em seguida, a consulta SQL/XML final é executada no banco de dados e seu resultado XML é apresentado ao usuário.

O algoritmo **GeraSQL/XML**, descrito na Listagem 5.2, gera uma consulta SQL/XML a partir de uma *Tree Pattern Query*. Para cada nó da árvore TPQ, o algoritmo considera dois casos: (Caso 1) se o nó não tem Nó-Resultado filho, o algoritmo obtém, dos TCs, a subconsulta SQL/XML correspondente ao elemento/atributo que rotula o nó. Em seguida, o algoritmo analisa os filhos do nó, verificando a existência de expressões condicionais (predicados ou filtros). Caso exista(m), ela(s) é/são traduzida(s) em um conjunto de expressões condicionais SQL (algoritmo **GeraFiltroSQL/XML** – Listagem 5.3); (Caso 2) Se o nó tem Nó(s)-Resultado filho, o algoritmo seleciona a subconsulta SQL/XML dos TCs, necessária para construir o conteúdo do elemento/atributo que rotula esse nó e, recursivamente, analisa cada Nó-Resultado filho (próximo nível de Nó-Resultado) para incrementar esse conteúdo.

No algoritmo **GeraSQL/XML**, a função **BuscaTC** recebe como parâmetro um tipo complexo e o caminho desse tipo, e obtém a subconsulta SQL/XML dos Templates de Consulta que corresponde ao tipo complexo de entrada. O algoritmo **GeraFiltroSQL/XML** é responsável pela tradução das expressões condicionais encontradas na árvore TPQ. A função recebe como parâmetro um conjunto de expressões condicionais e as traduzem em um conjunto de expressões condicionais SQL. Os casos de tradução de predicados são apresentados no algoritmo.

Entrada: nó de saída *$node*, tipo XML T e um caminho δ.
Saída: subconsulta SQL/XML.

```
1.  Seja: Qs uma subconsulta SQL/XML;
2.  Caso 1: Se $node não tem Nó-Resultado filho então
3.      Qs := BuscaTC( T , δ );
4.      Qs := Qs + GeraFiltroSQL/XML (ƒ ), onde ƒ é o conjunto de expressões
5.          condicionais filho de $node;
6.  Caso 2: Se $node tem Nó-Resultado filho então
7.      Qs := BuscaTC ( T , δ* );
8.      Qs := Qs + GeraFiltroSQL/XML (ƒ ), onde ƒ é o conjunto de expressões condicionais
9.          filho de $node;
10.
11.     //Análise dos nós que representam atributos
12.     Para cada Nó-Resultado filho faça
13.         Se $node tem n atributos então
14.             Seja: δi o caminho de $node para o nó atributo $ai de $node,
15.                 com 0<i<=n;
16.             attrQs := attrQs + "XMLAttributes(BuscaTC ( T, δ1 ) , ..., BuscaTC ( T,
17.             δn ) )";
18.         Fim Se
19.     Fim Para
20.
21.     subQs := "";
22.     Seja T' o tipo XML do elemento representado por $node;
23.     Para cada Nó-Resultado filho $e de $node, onde ¥ é o caminho de $node para $e faça
24.         subQs := subQs + GeraSQL/XML( $e , T', ¥ );
25.     Fim Para
26.
27.     Substitua %content% em Qs por attrQs+subQs;
28.
29. Retorne Qs;
```

Nota:
(ii) **BuscaTC** é uma função que analisa a *tabela template* de nome T nos TCs e retorna a subconsulta SQL/XML onde PATH="δ".

Listagem 5.2: Algoritmo **GeraSQL/XML**.

Entrada: conjunto de expressões condicional $f_1, \ldots, f_k$, onde $f_i = (e_1/.../e_n\ \theta\ v)$ e $n \geq 1$, $e_1/.../e_n$ é uma expressão de caminho, θ é um operador e v é um valor escalar.
Saída: Expressão SQL Π.

```
1.   Seja r uma alias para tabela R.
2.   Seja Π uma string.
3.
4.   Para cada expressão condicional f_i, 1≤i≤k faça
5.       Seja δ o caminho relacional e1/.../en.
6.           Caso { δ = a }:
7.               Π := Π + "( r.a φ 'v' )", onde φ é um operador SQL equivalente a θ;
8.           Caso { δ = a1,...,ak }:
9.               Π := Π + "( (r.a1 φ 'v' ) OR ... OR (r.ak φ 'v' ) )", onde φ é o operador SQL equivalente a θ;
10.          Caso { δ = φ1. ... .φm.a }, onde Ri é uma tabela, φ1 é um link de R para R1 e φi é um link de Ri-1
11.          para Ri, 2≤i≤m:
12.              Π := Π + "EXISTS (SELECT * FROM Joinφ(r) AND (rn.a φ 'v' ) )" , onde φ é um operador
13.                      SQL equivalente a θ;
14.          Caso { δ = φ1. ... .φm.{a1,...,ak } } onde Ri é uma tabela, φ1 é um link de R para R1 e φi é um link d
15.          Ri-1 para Ri, 2≤i≤m:
16.              Π := Π + "EXISTS (SELECT * FROM Joinφ(r) AND ( (rn.a1 φ 'v' ) OR ... OR (rn.ak φ 'v' ) ) )" ,
17.                      onde φ é um operador SQL equivalente a θ;
18.  Fim Para
19.
20.  Retorne Π;
```

Nota:
(i) **Joinφ**(*r*) é definida pelos seguintes fragmentos SQL:
*R*1 *r*1,..., *R*n *r*n
WHERE $r.a_k^{\ell 1} = r_1.b_k^{\ell 1}$, $1 \leq k \leq m_1$, AND $r_{i-1}.a_k^{\ell i} = r_i.b_k^{\ell i}$, $1 \leq k \leq m_i$, $2 \leq i \leq n$
Tal que, dado uma tupla **r** de **R**, então **r.**φ = SELECT **r**n FROM **Joinφ**(**r**).

Listagem 5.3: Algoritmo **GeraFiltroSQL/XML**.

5.4 Exemplo de Tradução de uma Consulta XQuery em uma Consulta SQL/XML

Considere, por exemplo, o esquema da visão Pedidos_XML apresentado na Figura 2.3. Considere a consulta XQuery Qx apresentada na Figura 5.3, a qual é definida sobre a visão XML Pedidos_XML. Qx retorna todos os pedidos dos clientes da cidade de Baltimore e, para cada pedido, são retornados somente os itens do pedido que possuam quantidade menor que 20.

A geração da TPQ Tx a partir da XQuery Qx é realizada nos seguintes passos:

```
for $v1 in view(Pedidos_XML)/Pedido
where $v1/Cliente/Endereco/Cidade = "Baltimore"
return <Pedido>
            {$v1/DataPedido,
             $v1/Cliente,
             for $v2 in $v1/Item
             where $v2/Quantidade < 20
             return <Item>
                        {$v2/Produto,
                         $v2/Quantidade}
                    </Item>}
        </Pedido>
```

Figura 5.3: Consulta XQuery Qx.

Passo 1: Análise da cláusula FOR da consulta XQuery. O algoritmo **GeraTPQX** analisa a cláusula FOR de Qx para obter o elemento raiz da TPQ (linha 8). A Figura 5.4(a) apresenta nó raiz Pedido de Tx.

Passo 2: Análise da cláusula WHERE da consulta XQuery. O algoritmo (linhas 11 – 27) analisa cada predicado da cláusula WHERE de Qx. Para o predicado **$v1/Cliente/Endereco/Cidade = "Baltimore"**, o elemento Cliente é adicionado na TPQ como nó filho do nó Pedido, o elemento Endereco é adicionado como nó filho do nó Cliente, o elemento Cidade é adicionado como nó filho do nó Endereco e o valor "Baltimore" é adicionado como nó filho do nó Cidade. Por fim, o operador = é adicionado à aresta entre os dois últimos nós. A Figura 5.4(b) mostra a TPQ Tx gerada após este passo.

Passo 3: Análise da cláusula RETURN da consulta XQuery. Finalmente, **GeraTPQX** (linhas 30 – 43) verifica cada elemento da cláusula RETURN de Qx. O elemento Pedido entra no caso 1 (linha 31), e, logo, um Nó-Resultado multivalorado com o nome do elemento é adicionado à árvore. O elemento DataPedido entra no caso *default* (linha 35), sendo adicionado na árvore como Nó-Resultado. O elemento Cliente entra no caso 1 (linha 31), sendo marcado na árvore como Nó-Resultado. A subconsulta "**for $v2 in $v1/Item...**" entra no caso 2 (linha 33) e **GeraTPQX** é chamado recursivamente para a construção da TPQ correspondente. A Figura 5.4(c) mostra a TPQ Tx gerada depois após esse passo.

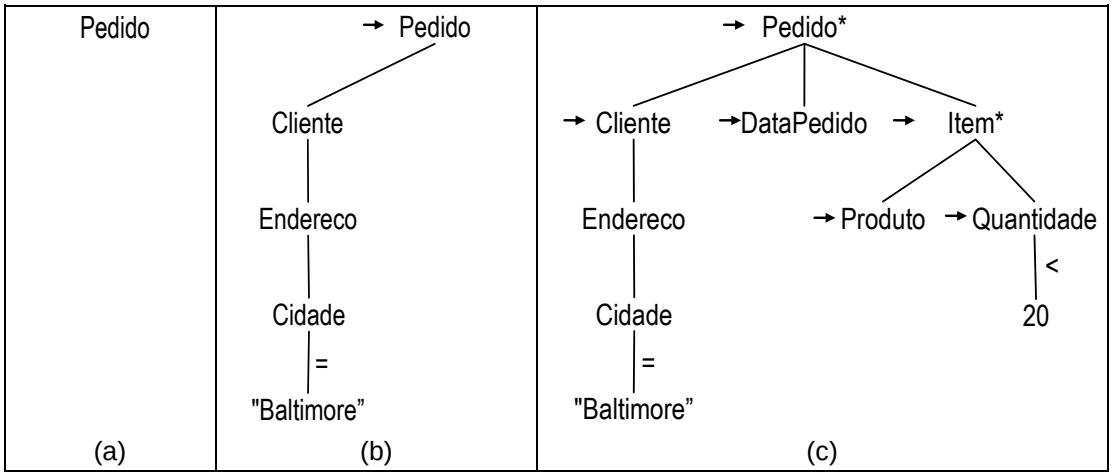


Figura 5.4: (a) TPQ gerada após a análise da cláusula FOR. (b) TPQ gerada após a análise da cláusula WHERE. (c) TPQ gerada após a análise da cláusula RETURN.

A geração da consulta SQL/XML Qs a partir da TPQ Tx é realizada nos passos:

Passo 1: Análise do nó raiz de Tx e tradução da expressão condicional em uma cláusula SQL equivalente. Para o nó raiz, o algoritmo **GeraSQL/XML** usa a *tabela template* correspondente ao tipo complexo do nó (TPedido_XML) e obtém a subconsulta SQL/XML apresentada na Figura 5.5 (Caso 2 - linha 6). A expressão condicional **$v1/Cliente/Endereco/Cidade = "Baltimore"**, da TPQ Tx, é traduzida em uma cláusula SQL correspondente, a qual é adicionada à cláusula WHERE da consulta SQL/XML Qs.

```
SELECT XMLELEMENT("Pedido", %content% )
FROM PEDIDOS_REL P, CLIENTES_REL C
WHERE P.PCLIENTE = C.CCODIGO AND C.CIDADE = “Baltimore”.
```

Figura 5.5: Consulta SQL/XML gerada no passo 1.

Passo 2: Geração das subconsultas SQL/XML para os Nós-Resultado filhos. Seguindo o algoritmo **GeraSQL/XML**, para cada Nó-Resultado filho são geradas uma ou mais subconsultas SQL/XML. O nó Cliente que não possui Nó-Resultado filho, entra no Caso 1 (linha 2) do algoritmo e é gerada a subconsulta SQL/XML apresentada na Figura 5.6. Este processo ocorre de forma similar para o elemento Data_pedido como mostra as Figura 5.7. O nó Item possui Nós-Resultado filho, então é gerada a subconsulta SQL/XML apresentada na Figura 5.8. Em seguida, o algoritmo é chamado recursivamente para processar os Nós-Resultado filho do nó Item e o *placeholder* %content% de Item é substituído pelas subconsultas SQL/XML criadas. Por fim, o

algoritmo substituirá o *placeholder* %content% de Qs pelas subconsultas SQL/XML criadas. A Figura 5.9 mostra a consulta SQL/XML Qs e, para cada fragmento SQL/XML da consulta, destaca a *tabela template* e o caminho de onde a mesma foi obtida.

Uma vez traduzida a consulta, o componente *Query Translator* envia a consulta SQL/XML Qs ao SGBD, que, por sua vez, gera o documento XML que é enviado para o usuário.

```
(SELECT XMLELEMENT("Cliente",
        XMLFOREST(C.cnome AS "Nome"),
        XMLELEMENT("Endereco", … ),
        XMLFOREST(C.cfone1 AS "Fone",
                  C.cfone2 AS "Fone",
                  C.cfone3 AS "Fone") )
FROM CLIENTES_REL C
WHERE C.CCODIGO = P.PCLIENTE)
```

Figura 5.6: Subconsulta SQL/XML correspondente ao nó Cliente.

```
XMLFOREST(P.Pdata_pedido AS "DataPedido")
```

Figura 5.7: Subconsulta SQL/XML correspondente ao nó DataPedido.

```
(SELECT XMLAGG( XMLELEMENT("Item", %content% ) )
 FROM ITENS_REL I
 WHERE I.IPEDIDO = P.PCODIGO AND I.QUANTIDADE < 20) )
```

Figura 5.8: Subconsulta SQL/XML correspondente ao nó Item.

```
SELECT XMLELEMENT("Pedido",
   XMLFOREST(P.Pdata_pedido AS "DataPedido"),                    TEMPLATE: TPedido_XML, CAMINHO = "DataPedido"
   (SELECT XMLELEMENT("Cliente",                                 TEMPLATE: TPedido_XML, CAMINHO = "Cliente "
      XMLFOREST(C.Cnome AS "Nome"),
         XMLELEMENT("Endereco",
            XMLFOREST(C.Crua AS "Rua"),
            XMLFOREST(C.Ccidade AS "Cidade"),
            XMLFOREST(C.Cestado AS "Estado"),
            XMLFOREST(C.Ccep AS "CEP") ),
        XMLFOREST(C.CFONE1 AS "Fone",
                  C.CFONE2 AS "Fone",
                  C.CFONE3 AS "Fone") )
    FROM CLIENTES_REL C
    WHERE C.CCODIGO = P.PCLIENTE),
   (SELECT XMLAGG( XMLELEMENT("Item",                            TEMPLATE: TPedido_XML, CAMINHO = "Item*"
      (SELECT XMLFOREST(D.PNOME AS "Produto"...)                 TEMPLATE: TItem_XML Template, CAMINHO = "Produto"
       FROM PRODUTOS_REL D
       WHERE D.PCODIGO = I.IPRODUTO),
      XMLFOREST(I.QUANTIDADE AS "Quantidade") ) )                TEMPLATE: TItem_XML Template, CAMINHO = "Quantidade"
    FROM ITENS_REL I
    WHERE I.IPEDIDO = P.PCODIGO AND I.QUANTIDADE < 20) )
FROM PEDIDOS_REL P, CLIENTES_REL C
WHERE P.PCLIENTE = C.CCODIGO AND C.CIDADE = “Baltimore”;
```

Figura 5.9: Consulta SQL/XML Qs.

5.5 Ferramenta *XQuery Browser*

No ***RelP***, o usuário pode formular graficamente uma consulta XQuery usando a ferramenta *XQuery Browser*. Esta ferramenta permite ao usuário que não domina a linguagem XQuery construir intuitivamente consultas sobre as visões. Considere os passos a seguir para consultar uma visão XML utilizando a *XQuery Browser*:

Passo 1: **Seleção da visão XML**. Via interface gráfica, o usuário seleciona uma visão XML publicada no ***RelP***. A Figura 5.10 apresenta a interface de seleção de visões XML do *XQuery Browser*. Em ①, a visão DEPARTAMENTOS_REL está selecionada, e em ② é apresentado seu *XML Schema*. O arquivo de capacidades do serviço *WXService* é apresentado em ③.

Passo 2: **Construção da Consulta**. Após a escolha da visão XML, é gerada a árvore que representa o esquema da visão. Em seguida, o usuário seleciona na árvore os elementos que farão parte do resultado da consulta e, se desejável, define expressões condicionais (filtro). A Figura 5.11 apresenta a interface gráfica principal da ferramenta

XQuery Browser. Em ④, temos o esquema da visão DEPARTAMENTOS_REL em forma de árvore. De acordo com os atributos selecionados e a expressão condicional definida em ④, a consulta consiste em retornar o nome, a data de fundação e os nomes e as ruas de todos dos empregados de todos os departamentos.

Passo 3: **Construção da Consulta XQuery e resultado XML**. *XQuery Browser* constrói a consulta XQuery a partir dos elementos selecionados em ④, e, através do serviço *WXService*, a envia para o componente *Query Translator*, o qual realiza a tradução da consulta e retorna o resultado ao usuário. Na Figura 5.11, em ⑤ a consulta XQuery gerada, em ⑥ temos a TPQ gerada e em ⑦ o resultado XML retornado para o usuário.

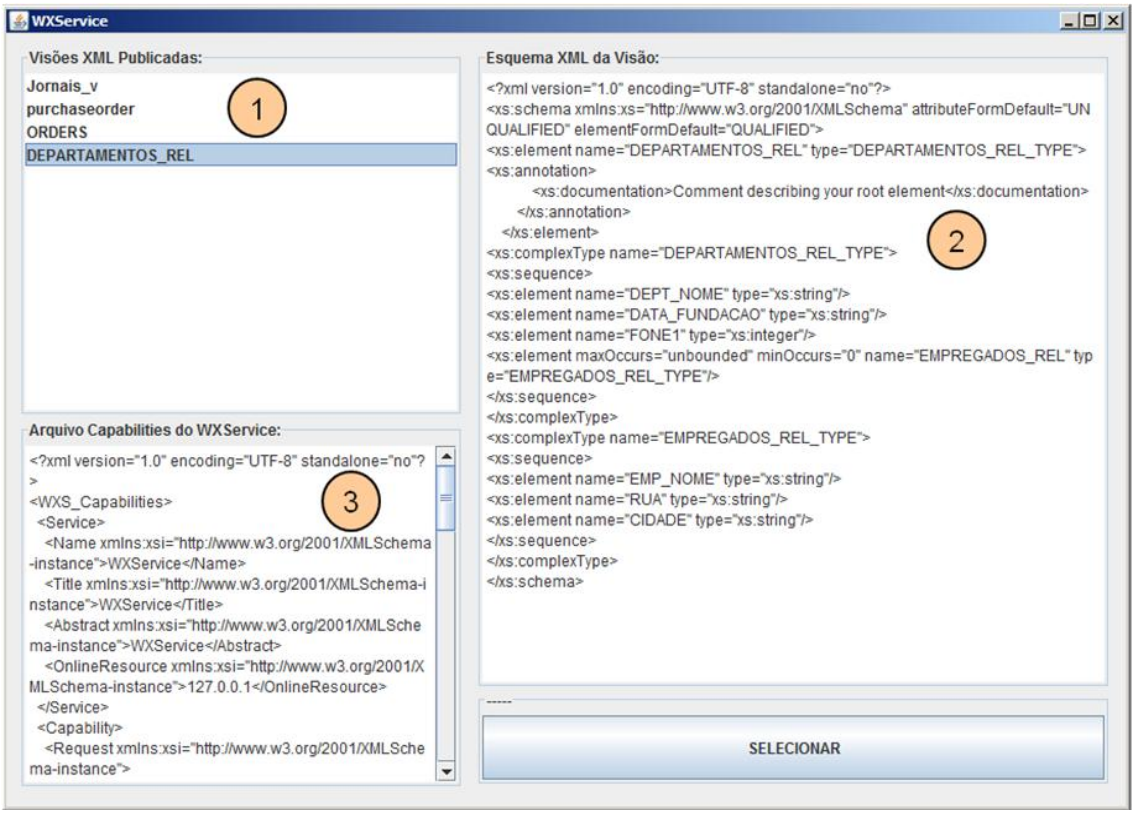


Figura 5.10:Interface de seleção de visões XML da *XQuery Browser*.

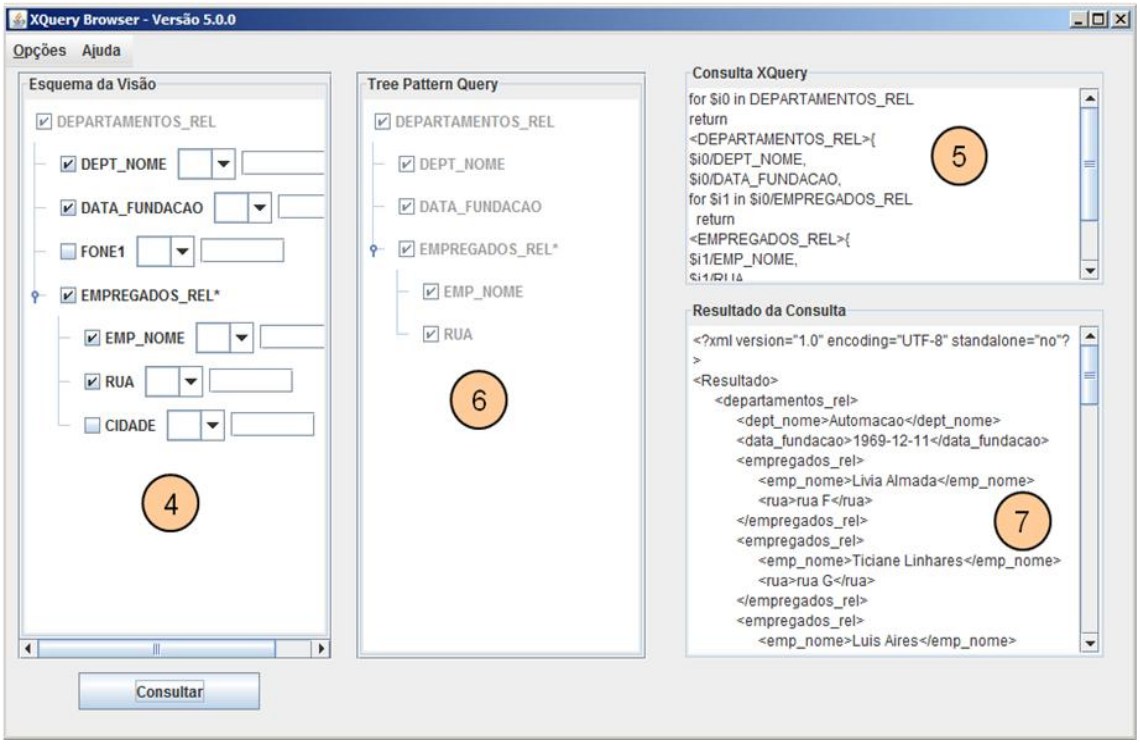


Figura 5.11:Interface principal do *XQuery Browser*.

5.6 Conclusão

Neste capítulo, apresentamos em detalhes o processo de tradução de consultas no ***RelP***, o qual é realizado em dois passos: (i) uma *Tree Pattern Query* (TPQ) é gerada a partir da consulta XQuery e, em seguida, (ii) uma consulta SQL/XML é gerada a partir da TPQ. Para tanto, mostramos o algoritmo **GeraTPQX** que gera uma TPQ a partir de uma consulta XQuery e o algoritmo **GeraSQL/XML** que gera uma consulta SQL/XML a partir de uma TPQ. Em nossa abordagem, o processo de tradução de consultas utiliza as TPQs para possibilitar a restrição do escopo da consulta XQuery e para facilitar a geração da consulta SQL/XML. Além disso, é utilizado os Templates de Consulta para simplificar e garantir a eficiência.

Discutimos também o processo de consulta de uma visão através da *XQuery Browser*. A ferramenta oferece uma interface gráfica para apoiar a construção de consultas XQuery. Finalmente, mostramos um exemplo do processo de construção de uma consulta XQuery usando o *XQuery Browser*.

Capítulo 6
Adaptação do *RelP* à especificação *Web Feature Service*

*Neste Capítulo, apresentamos uma adaptação do **RelP** à especificação Web Feature Service (WFS), de modo a permitir a publicação e consulta de visões GML sobre base de dados relacional. O Capítulo está organizado da seguinte forma. A Seção 6.1 apresenta uma introdução à especificação WFS. A Seção 6.2 apresenta o processo de publicação de visões GML no **RelP**. A Seção 6.3 mostra como consultas WFS são traduzidas em consultas SQL/XML no **RelP**. Finalmente, a Seção 6.4 apresenta a conclusão.*

6.1 Introdução

Durante a última década, a troca de informações entre as organizações tornou-se não somente possível, mas primordial. No caso dos dados espaciais, essa disseminação requer cuidados especiais, pois estes dados são bem mais complexos do que os dados convencionais com respeito à sintaxe e à semântica.

A complexidade e a riqueza dos modelos de dados geográficos proporcionam diferentes formas de representação das informações espaciais. Além disso, os dados espaciais encontram-se armazenados não somente em bancos de dados geográficos, mas também podem estar armazenados em banco de dados relacionais ou objeto-relacionais, arquivos-texto, planilhas eletrônicas, formatos proprietários dos Sistemas de Informações Geográficas (SIGs), etc. Assim sendo, realizar o intercâmbio de dados espaciais é uma tarefa bastante complexa, devido à diversidade de aplicações existentes e ao fato de que os dados espaciais apresentam diferentes sintaxes, semânticas, representações e formatos.

Neste contexto, criou-se o Consórcio OpenGIS (*Open Geospatial Consortium* – OGC) com o objetivo de promover o desenvolvimento de tecnologias que facilitem a interoperabilidade entre SIGs. O OGC define duas importantes iniciativas: a linguagem *Geography Markup Language* (GML) e a especificação *Web Feature Service* (WFS).

A linguagem GML é uma linguagem baseada em XML considerada o formato padrão para representação de dados geográficos na Web. O objetivo da GML é oferecer um conjunto de construções básicas, incluindo o modelo de *features*[1] geográficas e uma coleção de meta-classes de objetos geográficos, com os quais um usuário pode estruturar e descrever seus dados georeferenciados [76]. Apesar de algumas limitações, a linguagem GML representa um avanço significativo para prover a interoperabilidade entre aplicações geográficas.

A especificação WFS, por sua vez, é considerada um formato padrão para troca de dados geográficos na Web. A especificação define um conjunto de interfaces e operações para a manipulação e acesso a dados geográficos em GML. As principais operações definidas pela WFS são: (i) getCapabilities: permite ao cliente solicitar e recuperar a descrição dos metadados ou arquivo de capacidades, que são os documentos que descrevem as habilidades da implementação específica do servidor; (ii) describeFeatureType: permite ao cliente recuperar a descrição da estrutura de qualquer visão GML (um esquema XML); (iii) getFeature: permite ao cliente recuperar instâncias de uma visão GML.

Com base nessas iniciativas, propomos uma adaptação do nosso *framework* à especificação WFS, de modo a permitir a publicação e consulta de visões GML sobre base de dados relacional. A principal modificação no ***RelP*** reside sobre o serviço Web *WXService*. Uma vez que a especificação WFS define um conjunto de métodos e uma linguagem de consulta própria, foi necessário implementar um novo serviço Web, chamado *WFService*, o qual é uma adaptação do serviço *WXService* com base na especificação WFS. O serviço Web *WFService* implementa os seguintes métodos da especificação WFS: *getCapabilities*, *describeFeatureType* e *getFeature*.

As principais modificações foram,

i. Adaptação dos algoritmos de publicação de visões XML para permitir a publicação de visões de dados geográficos (visões GML);
ii. Adaptação dos algoritmos de tradução de consulta para traduzir consultas WFS sobre visões GML em consultas SQL/XML sobre o banco de dados; e
iii. Modificação das ferramentas *XML View Publisher*, para tratar visões GML, e *XQuery Browser*, para criar consultas WFS de forma gráfica.

[1] Uma feição ou *feature* é uma abstração de um fenômeno do mundo real e está associada a uma localização relativa à terra.

Neste Capítulo, apresentamos na Seção 6.2 o processo de publicação de visões GML no ***RelP***. Na Seção 6.3, mostramos o processo de tradução de consultas WFS em consultas SQL/XML. E, na Seção 6.4, apresentamos a conclusão.

6.2 Publicando visões GML no *RelP*

No ***RelP***, uma visão GML é também especificada por um esquema XML e um conjunto de Assertivas de Correspondência (ACs). Uma visão GML é representada por uma quádrupla $F = \langle e, T_e, R, \mathcal{A} \rangle$, onde e é o nome do elemento primário da visão, T_e é o tipo GML do elemento e, R é o esquema da tabela pivô, e $\mathcal{A}$ é o conjunto de ACs de F, as quais definem os mapeamentos entre o esquema da visão GML e o esquema do banco de dados. A tabela pivô é também chamada de *tabela de feição*, pois ela contém pelo menos um atributo geométrico.

No ***RelP***, os passos para publicação de uma visão GML são semelhantes aos passos de publicação de uma visão XML, a diferença está somente no tratamento dos dados geográficos. Dessa forma, no algoritmo **GeraSubSQL/XML** (vide Listagem 4.2 do Capítulo 4), foi adicionado o caso 3 (linha 6) que trata o uso de atributos geométricos. Como mostra o caso 3 do algoritmo **GeraSubSQL/XML**, *g* é uma função (proprietária do SGBD) que recebe um atributo do tipo geométrico e gera o fragmento GML correspondente. O Oracle, por exemplo, provê a função SDO_UTIL.TO_GMLGEOMETRY(), a qual gera elementos GML a partir do tipo geométrico proprietário SDO_GEOMETRY. No PostgreSQL, a extensão PostGIS implementa a função AsGML() para tal fim.

Exemplo 6.1

Por exemplo, considere o esquema relacional do banco de dados Postos apresentado na Figura 6.1 e considere que o usuário deseje publicar uma visão GML F_Posto sobre o esquema relacional Postos. A relação Posto_rel é denominada tabela pivô por possuir relacionamento 1-1 com a visão e por possuir o elemento geométrico.

Para a *publicação semi-automática* da visão F_Posto com a ferramenta *XML View Publisher*, considere os passos a seguir:

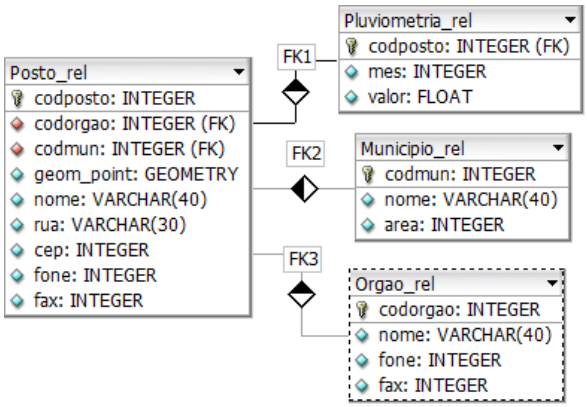


Figura 6.1: Esquema do Banco Postos.

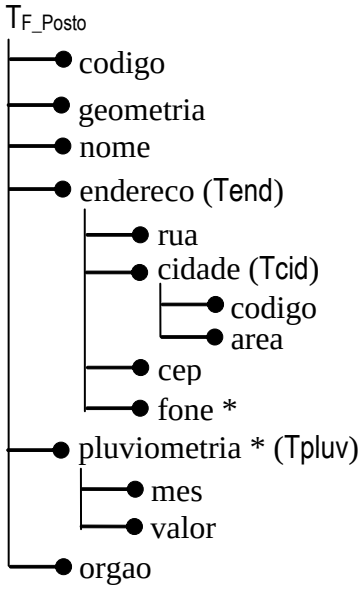


Figura 6.2: Esquema da Visão GML F_Posto.

Passo 1: **Definição do esquema da visão GML**. O usuário define através de uma interface gráfica, o esquema da visão GML com base no esquema relacional Postos. Posto_rel é uma tabela relacional, cujo dado geográfico está representado pelo atributo geométrico geom_point. A Figura 6.2 mostra o esquema da visão F_Posto, cujo elemento primário é F_Posto do tipo TF_Posto.

Passo 2: **Definição das Assertivas de Correspondência da visão GML**. O usuário define graficamente o mapeamento entre o esquema relacional e o esquema GML da visão, gerando as Assertivas de Correspondência da visão, como mostrado na Figura 6.3. As assertivas são geradas relacionando-se os elementos e atributos de TF_Posto com os atributos/caminhos de Posto_rel. Por exemplo, a assertiva φ_2:[TF_Posto/geometria ≡ Posto_rel/geom_point] relaciona o elemento geometria de TF_Posto da visão com o atributo geom_point da relação Posto_rel.

Passo 3: **Geração dos Templates de Consulta da visão GML**. Com base nas ACs, a XVP gera os Templates de Consulta (TCs) da visão GML apresentados na Figura 6.4. Em seguida, o esquema os TCs da visão são registrados e armazenados no ***RelP*** via

componente *Catalogue*. Os TCs são gerados através do algoritmo **GeraTC** apresentado na Seção 4.2 do Capítulo 4.

Suponha que o banco de dados Postos foi criado utilizando o SGBD Oracle. Na geração dos Templates de consulta da visão F_Posto, a seguinte subconsulta SQL/XML foi gerada para o elemento geometria da visão:

XMLFOREST(**SDO_UTIL.TO_GMLGEOMETRY**("P.GEOM_POINT") AS "geometria").

Note que, neste caso, a função TO_GMLGEOMETRY do Oracle será responsável por gerar os valores GML a partir do atributo geometria.

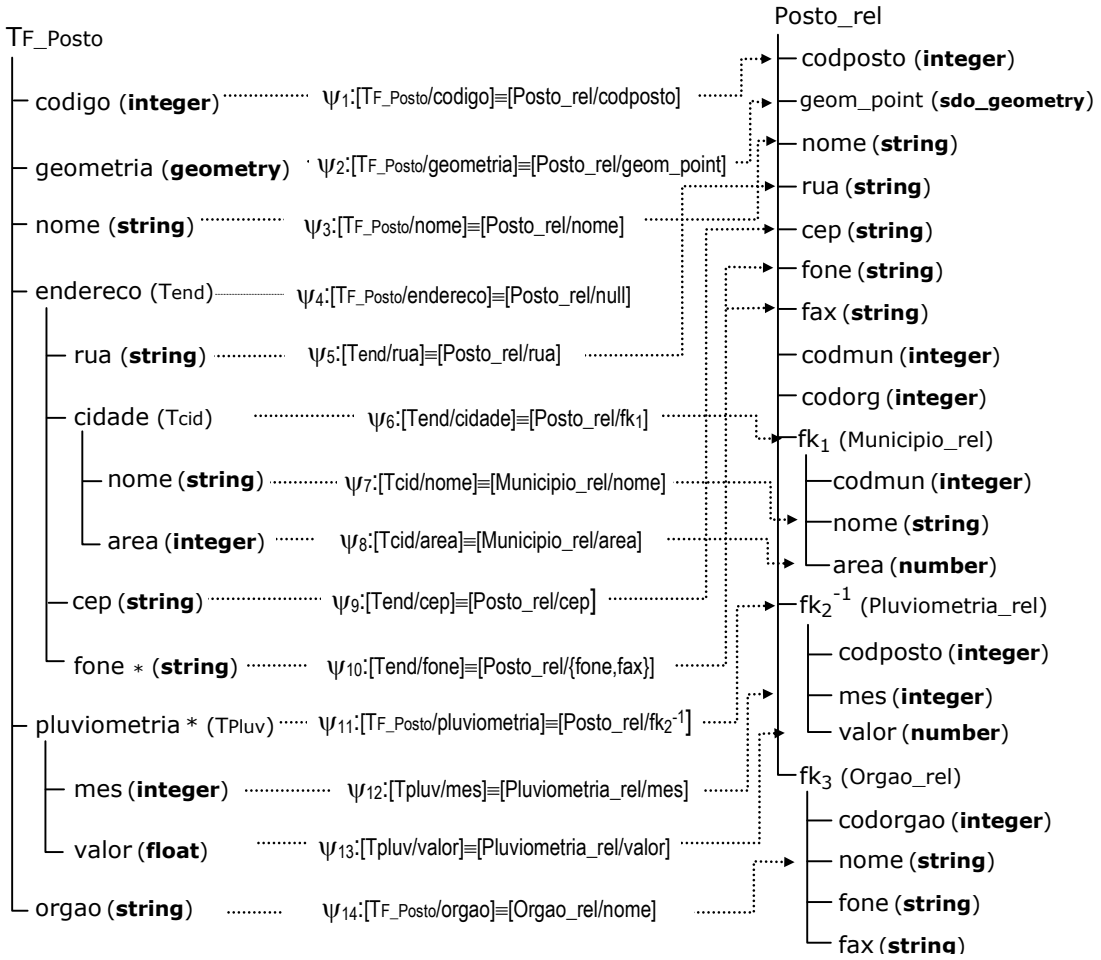


Figura 6.3: Assertivas de Correspondências da visão F_Posto.

Template TF_Posto

CAMINHO	SUBCONSULTA SQL/XML
F_Posto*	SELECT XMLELEMENT("gml:featureMember", XMLELEMENT("F_Posto", %content%) FROM POSTO_REL P
...	...
codigo	XMLForest(P.CODPOSTO AS "codigo")
geometria	XMLFOREST(SDO_UTIL.TO_GMLGEOMETRY("P.GEOM_POINT") AS "geometria")
nome	XMLForest(P.NOME AS "nome")
Endereco*	(SELECT XMLELEMENT("Endereco", %content%))
Endereco	(SELECT XMLELEMENT("Endereco", XMLFOREST(P.RUA AS "Rua"), (SELECT XMLELEMENT("Cidade", XMLFOREST(M.NOME AS "Nome"), XMLFOREST(M.AREA AS "Area") FROM MUNICIPIO_REL M WHERE M.CODMUN = P.CODMUN), XMLFOREST(P.CEP AS "Cep") XMLFOREST(P.FONE AS "Fone", P.FAX AS "Fax")))
...	...
Pluviometria*	(SELECT XMLAGG(XMLELEMENT("Pluviometria", %content%) FROM PLUVIOMETRIA_REL PL WHERE PL.CODPOSTO = P.CODPOSTO)
Pluviometria	(SELECT XMLAGG(XMLELEMENT("Pluviometria", XMLFOREST(PL.MES AS "Mes"), XMLFOREST(PL.VALOR AS "Valor"))) FROM PLUVIOMETRIA_REL PL WHERE PL.CODPOSTO = P.CODPOSTO)
...	...
Orgao	(SELECT XMLELEMENT("Orgao" AS O.O.NOME) FROM ORGAO_REL O WHERE O.CODORGAO = P.CODORGAO)
...	...

Template TEnd

CAMINHO	SUBCONSULTA SQL/XML
Rua	XMLFOREST(P.RUA AS "Rua")
Cidade*	XMLELEMENT("Cidade", %content%)
Cidade/Nome	XMLFOREST(M.NOME AS "Nome")
Cidade/Area	XMLFOREST(M.AREA AS "Area")
...	...

Template TCid

CAMINHO	SUBCONSULTA SQL/XML
Nome	XMLFOREST(M.NOME AS "Codigo")
Area	XMLFOREST(M.AREA AS "Area")

Template TPluv

CAMINHO	SUBCONSULTA SQL/XML
Mes	XMLFOREST("Mes", PL.MES)
Valor	XMLFOREST("Valor", PL.VALOR)

Figura 6.4: Templates de Consulta da visão F_Posto.

6.3 Consultando visões GML no *RelP*

A especificação WFS define uma linguagem própria para consultar dados GML. Uma consulta escrita nessa linguagem (consulta WFS) é enviada ao servidor WFS através da operação getFeature.

Segundo a especificação WFS, uma operação getFeature contém uma ou mais consultas WFS, cada uma encapsulada em um elemento XML Query. O elemento Query, por sua vez, (i) possui um atributo TypeName, o qual define o nome da visão GML sobre a qual a consulta é feita, (ii) possui zero ou mais elementos PropertyName, os quais especificam os elementos da visão que farão parte do resultado da consulta, e (iii) pode possuir um elemento filter, o qual define o(s) filtro(s) da consulta. Uma consulta WFS sempre retorna uma instância do tipo da visão, de modo que o usuário pode escolher quais subelementos da visão aparecerão nas instâncias retornadas, mas não pode reestruturar os elementos ou criar novos a partir dos existentes. Maiores detalhes sobre a especificação WFS e suas operações podem ser encontrados em [38].

A Figura 6.5 mostra o exemplo de uma consulta WFS definida sobre a visão GML F_Posto (Figura 6.2). A consulta recupera o nome, geometria, endereço da cidade e pluviometrias dos postos da Universidade Federal do Ceará.

```
<wfs:GetFeature outputFormat="GML2" …>
  <wfs:Query typeName="F_Posto">
       <wfs:PropertyName> nome </wfs:PropertyName>
       <wfs:PropertyName> geometria </wfs:PropertyName>
       <wfs:PropertyName> endereco/cidade </wfs:PropertyName>
       <wfs:PropertyName> pluviometria </wfs:PropertyName>
        <ogc:Filter>
                 <ogc:PropertyIsEqualTo>
                       <ogc:PropertyName> orgao </ogc:PropertyName>
                         <ogc:Literal> Univerisadade Federal do Ceara </ogc:Literal>
                   </ogc:PropertyIsEqualTo>
       </ogc:Filter>
  </wfs:Query>
```

Figura 6.5: Exemplo de uma consulta WFS.

No ***RelP***, o componente *Query Translator* do serviço Web *WFService* implementa o método getFeature da especificação WFS. O componente traduz uma consulta WFS Qf definida sobre o esquema de uma visão GML em uma consulta SQL/XML Qs sobre o esquema do banco de dados relacional. Qs é uma tradução correta de Qf, o que significa que o resultado da execução de Qf sobre a materialização da visão GML é equivalente ao resultado da execução de Qs sobre o banco de dados, em um dado instante.

No componente *Query Translator*, cada consulta do elemento Query definida sobre uma visão GML é traduzida em uma *Tree Pattern Query* (TPQ) e, em seguida, a TPQ é traduzida em uma consulta SQL/XML definida sobre o esquema do banco relacional. Os resultados das consultas são compilados em um único documento GML, o qual é enviado ao usuário. Nas seções a seguir, detalhamos os passos do processo de tradução de consultas WFS no ***RelP***.

6.3.1 Geração da *Tree Pattern Query*

Para traduzir uma consulta WFS Qw em uma TPQ Tw correspondente, implementamos o algoritmo **GeraTPQG**, apresentado na Listagem 6.1. Em resumo, o algoritmo recebe como parâmetro a consulta WFS e gera a TPQ correspondente em três etapas:

(iv) **Análise do Elemento QUERY**: o nó raiz da TPQ é obtido a partir do atributo typeName do elemento Query.

(v) **Análise do Elemento FILTER**: os predicados são analisados e adicionados à TPQ. O algoritmo verifica os subelementos do elemento Filter, analisando a expressão de caminho de cada predicado. Para cada elemento do caminho, é adicionado um nó correspondente na TPQ.

(vi) **Análise dos Elementos PROPERTYNAME**: Para cada elemento PropertyName, o algoritmo "marca" e/ou adiciona o nó na TPQ correspondente ao conteúdo deste elemento. Os nós marcados, chamados de Nós-Resultado, fazem parte do resultado da consulta. Nesta fase, para facilitar a tradução da TPQ na consulta SQL/XML correspondente, os nós que representam elementos multivalorados também são marcados com um *.

Entrada: Consulta WFS Qw
Saída: Tree Pattern Query

```
1.    Seja TPQ uma Tree Pattern Query;
2.    Seja P = (δ θ v) um predicado, onde:
3.          δ é uma expressão de caminho na forma e1/.../en, n≥1,
4.          θ é um operador,
5.          v é um valor escalar.
6.
7.    //Análise do Elemento QUERY
8.    TPQ.addRaiz( Qw/QUERY/typeName );
9.
10.   //Análise do elemento FILTER
11.   Para cada predicado P = (δ θ v) de Qw/FILTER faça
12.         Aux := TPQ.raiz();
13.         Para cada elemento e de δ = e1/.../en faça
14.               Se (e existe na TPQ) então
15.                     Aux := e;
16.               Senão
17.               Aux.addFilho(e);
18.               Aux := e;
19.               Fim Se
20.         FimPara
21.    Aux.addFilhoComArestaValorada( v , θ );
22.   FimPara
23.
24.   //Adição das operações entre os predicados.
25.   Para cada operador O entre predicados Pi e Pi+1 de Qx faça
26.   TPQ .addOperador ( O, Pi, Pi+1 );
27.   Fim Para
28.
29.   //Análise dos elementos PROPERTYNAME
30.   Para cada elemento E de Qx/PROPERTYNAME faça
31.         Caso 1: Se E existe na TPQ então
32.               TPQ.marcarComoNóResultado( E );
33.         Caso 2: Se E tem uma expressão de caminho então
34.               TPQ.addFilho (GeraTPQG ( E ));
35.         Default:
36.               TPQ.addFilho( E );
37.               TPQ.marcarComoNóResultado( E );
38.         Fim Caso;
39.         Se (E é o elemento raiz da TPQ) ou (E é um elemento multivalorado) então
40.               TPQ.marcarComoMultivalorado( E );
41.         Fim Se
42.   Fim Para
43.
44.   Retorne TPQ.
```

Nota:
(iii) **addFilhoComArestaValorada** é uma função que constrói um nó filho v com aresta valorada θ.

Listagem 6.1: Algoritmo **GeraTPQG**.

6.3.2 Geração da consulta SQL/XML

O próximo passo do processo de tradução de consulta consiste em traduzir a TPQ gerada a partir da consulta WFS em uma consulta SQL/XML. Para tanto, este processo utiliza o algoritmo **GeraSQL/XML** descrito na Listagem 5.2 do Capítulo 5.

Para o tratamento de filtros de consultas WFS, foi adicionado à função **GeraFiltroSQL/XML** (Listagem 5.3 do Capítulo 5) uma chamada à função **GeraFiltroWFS.** A função **GeraFiltroWFS**, apresentada na Listagem 6.2, retorna expressões condicionais SQL a partir do filtros da consulta WFS. No algoritmo, para cada predicado Ei do filtro da consulta WFS, a função **TraduzPredicado** gera uma expressão condicional SQL Pi, a qual será usada na cláusula WHERE da consulta SQL, e uma lista Li de nomes de tabelas requeridas para processar Pi, a qual será usada na cláusula FROM da consulta SQL.

Entrada: Elemento <filter> de uma consulta WFS F
Saída: Expressões SQL Π

```
1.  Seja: P uma expressão condicional SQL;
2.  Seja: L uma lista de nomes de esquemas requeridas para processar as condições P;
3.
4.  Para cada predicado Ei de F faca
5.      Π := Π + TraduzPredicado(Ei, Pi, Li )
6.
7.  Retorne Π;
```

Nota:
(i) TraduzPredicado(E, P, L) é uma função que gera uma expressão SQL Π (P , L) a partir do predicado E.

Listagem 6.2: Algoritmo **GeraFiltroWFS**.

A Tabela 6.1 resume a função **TraduzPredicado** adaptada ao SGBD Oracle. Em síntese, esta função identifica o operador do predicado <E>, especificado na coluna esquerda da tabela, e o traduz em uma expressão SQL de acordo com algoritmos/expressões definidos na coluna direita da tabela. Para a função considere:

- gml2sql(elemento <geometry>): função que traduz o elemento <geometry> em um tipo proprietário do SGBD. Neste caso, usaremos o tipo geométrico do SGBD Oracle, mdsys.sdo_geometry. Todavia, a adaptação para um outra função é simples.
- mask($\mathcal{E}$): função que retorna o relacionamento topológico de acordo com $\mathcal{E}$.

- symbol($\mathcal{E}$): função que retorna o operador de comparação de acordo com $\mathcal{E}$.
- parse(value, T): função que retorna o valor de value de acordo com o tipo T.
- logicSymbol($\mathcal{E}$): função que retorna o operador lógico de acordo com <e>.
- arithmeticSymbol ($\mathcal{E}$): função que retorna o operador aritmético de acordo com $\mathcal{E}$.

<$\mathcal{E}$> é um operador espacial da forma:

<$\mathcal{E}$> <PropertyName> T/c</PropertyName> <geometry>…</geometry> </$\mathcal{E}$>, onde <$\mathcal{E}$> é pode ser o elemento: <Equals>, <Disjoint>, <Touches>, <Within>, <Overlaps>, <Crosses>, <Intersects>, <BBox> ou <Contains>, onde [T/c] ≡ [*R*/a] e c é do tipo geométrico	$\mathcal{P}$ = $\mathcal{P}$ + "(mdsys.sdo_relate(" + *r*.a + ", " + gml2sql(<geometry>) + ", 'mask="+mask(<$\mathcal{E}$>)+" ') = 'TRUE')"
<DWithin> <PropertyName>T/c</PropertyName> <geometry>…</geometry> <distance>value</distance> </DWithin>, onde [T/c] ≡ [*R*/a] e c é do tipo geométrico	$\mathcal{P}$ = $\mathcal{P}$ + "(mdsys.sdo_within_distance(" + *r*.a + ", " + gml2sql(<geometry>) + ", 'distance = " + value + " ') = 'TRUE')"

<$\mathcal{E}$> é um operador de comparação da forma:

<$\mathcal{E}$> <Expression$_1$>…</Expression$_1$> <Expression$_2$>…</Expression$_2$> </$\mathcal{E}$>, onde <$\mathcal{E}$> pode ser o elemento: <PropertyIsEqualTo>, <PropertyIsNotEqualTo>, <PropertyIsLessThan>, <PropertyIsGreaterThan>, <PropertyIsLessThanOrEqualTo> ou <PropertyIsGreaterThanOrEqualTo>	$\mathcal{P}$=$\mathcal{P}$ + "(" + **TraduzPredicado**(<expression$_1$>,$\mathcal{P}$, $\mathcal{L}$) + symbol(<$\mathcal{E}$>) + **TraduzPredicado**(<expression$_2$>,$\mathcal{P}$, $\mathcal{L}$) + ")"
<PropertyIsLikeTo> <PropertyName> T/c </PropertyName> <Literal>value</Literal> </PropertyIsLikeTo>, onde [T/c] ≡ [*R*/δ] e T_a é o tipo de a	**Se** δ é da forma φ.a, então **Se** c tem uma ocorrência, então $\mathcal{P}$ = $\mathcal{P}$ + "*r*.a = r_1.a_1 and "+…+ " r_{i-1}.a_{i-1} = r_i.a_i and "+"(" + r_n.a + " like "%" + parse(value, T_a) + "%")", onde 1≤i≤n **Senao** /* varias ocorrencias $\mathcal{P}$ = $\mathcal{P}$ + "exist (select * from " + **Joinφ**(*r*) + " and (" + r_n.a + " like "%" + parse(value, T_a) + "%")" **FimSe** **Senão** /* não ha caminho $\mathcal{P}$ = $\mathcal{P}$ + "(" + *r*.a + " like "%" + parse(value, T_a) + "%")" **FimSe** **Se** δ é da forma φ.a, entao $\mathcal{L}$ ∪ "R_i r_i", onde 1 ≤ i ≤ n
<PropertyIsNull> <PropertyName> **T/c** </PropertyName> </PropertyIsNull>, onde [T/c] ≡ [*R*/δ] e T_a é o tipo de a	**Se** δ é da forma φ.a, então **Se** c tem uma ocorrência, então $\mathcal{P}$ = $\mathcal{P}$ + "*r*.a = r_1.a_1 and "+…+ " r_{i-1}.a_{i-1} = r_i.a_i and "+"(" + r_n.a + " is null)", onde

	1≤i≤n **Senao** /* varias ocorrencias $\mathcal{P}$ = $\mathcal{P}$ + "exist (select * from " + **Joinφ**(*r*) + " and (" + r_n.a + " is null) " **FimSe** **Senão** /* não ha caminho $\mathcal{P}$ = $\mathcal{P}$ + "(" + *r*.a + " is null)" **FimSe** **Se** δ é da forma φ.a, entao $\mathcal{L}$ ∪ "R_i r_i", onde 1 ≤ i ≤ n
<PropertyIsBetween> <Expression$_1$>…</Expression$_1$> <LowerBoundary> <Expression$_2$>…</Expression$_2$> </LowerBoundary> <UpperBoundary> <Expression$_3$>…</Expression$_3$> </UpperBoundary> </PropertyIsBetween>	"(" + **TraduzPredicado**(<expression$_1$>,$\mathcal{P}$, $\mathcal{L}$) + BETWEEN + **TraduzPredicado**(<expression$_2$>,$\mathcal{P}$, $\mathcal{L}$) + "AND" + **TraduzPredicado**(<expression$_3$>,$\mathcal{P}$, $\mathcal{L}$) + ")"

<$\mathcal{E}$> é uma operador logico da forma:

<$\mathcal{E}$> <operator$_1$>…</operator$_1$> <operator$_2$>…</operator$_2$> …. <operator$_n$>…</operator$_n$> </$\mathcal{E}$>, onde <$\mathcal{E}$> é um elemento <And> ou <Or> e o elemento <operator$_i$> pode ser um operador espacial, comparação ou lógico	"(" + **TraduzPredicado**(<operator$_1$>,$\mathcal{P}$, $\mathcal{L}$) + logicSymbol(<$\mathcal{E}$>) + **TraduzPredicado**(<operator$_2$>,$\mathcal{P}$, $\mathcal{L}$) +...+ logicSymbol(<$\mathcal{E}$>) + **TraduzPredicado**(<operator$_n$>,$\mathcal{P}$, $\mathcal{L}$) ")"
<Not> <operator$_1$>…</operator$_1$> </Not>, onde o elemento <operator$_1$> pode ser um operador espacial, comparação ou lógico	"(NOT " + **TraduzPredicado**(<operator$_1$>,$\mathcal{P}$, $\mathcal{L}$) + ")"

<$\mathcal{E}$> é um operador aritmético da forma:

<$\mathcal{E}$> <expression$_1$>…</expression $_1$> <expression $_2$>…</expression $_2$> </$\mathcal{E}$> onde <$\mathcal{E}$> pode ser o elemento: <Add>, <Sub>, <Mul> ou <Div>	"(" + **TraduzPredicado**(<expression$_1$>,$\mathcal{P}$, $\mathcal{L}$) + arithmeticSymbol(<$\mathcal{E}$>) + **TraduzPredicado**(<expression$_2$>,$\mathcal{P}$, $\mathcal{L}$) ")"

<$\mathcal{E}$> é um operador de identificação da forma:

<FeatureId fid="value"/>	*r*.k = value, onde k é o nome da coluna da tabela pivô que armazena o id da visão de feição

<$\mathcal{E}$> é uma expressão :

<Add>, <Sub>, <Mul>, <Div>	**TraduzPredicado**(<$\mathcal{E}$>,$\mathcal{P}$, $\mathcal{L}$)
<Literal>value</Literal>	parse(value, T)
<PropertyName> T/c </PropertyName> , onde [T/c] ≡ [*R*/δ] e T_a é o tipo de a	**Se** δ é da forma φ.a, então **Se** c tem uma ocorrência, então $\mathcal{P}$ = $\mathcal{P}$ + "*r*.a = r_1.a_1 and "+…+ " r_{i-1}.a_{i-1} = r_i.a_i and "+"(" + r_n.a + " is null)", onde 1≤i≤n

	Senao /* varias ocorrencias $\mathcal{P}$ = $\mathcal{P}$ + "exist (select * from " + **Join**φ(*r*) + " and (" + r_n.a + " is null) " **FimSe** **Senão** /* não ha caminho $\mathcal{P}$ = $\mathcal{P}$ + "(" + *r*.a + " is null)" **FimSe** **Se** δ é da forma φ.a, entao $\mathcal{L}$ ∪ "R_i r_i", onde 1 ≤ i ≤ n

Tabela 6.1: Função **TraduzPredicado**.

6.3.3 Exemplo de Tradução de uma consulta WFS em uma consulta SQL/XML

Considere, por exemplo, o esquema da visão GML F_Posto apresentado na Figura 6.2, a qual exporta dados do esquema relacional Postos (Figura 6.1). Considere também a consulta WFS Qw mostrada na Figura 6.6(a), a qual é definida sobre a visão GML *F_Posto*. Qw obtém o nome, endereço da cidade, pluviometrias e a geometria dos postos das cidades com área menor que 7000. A tradução da consulta Qw em uma consulta SQL/XML é feita nos passos a seguir:

Passo 1: Geração da *Tree Pattern Query*. Com base no algoritmo **GeraTPQG**, (i) a raiz da TPQ Tw é obtida a partir do atributo typeName do elemento Query. (ii) Em seguida, o algoritmo analisa o elemento Filter de Qw e, com base no caminho definido no elemento PropertyIsLessThan/PropertyName, são adicionados os nós endereço, cidade, área e 7000, e entre estes dois últimos uma aresta rotulada com o operador menor (<). (iii) Finalmente **GeraTPQG** analisa os elementos PropertyName de Qw e os rotula e/ou adiciona na TPQ como Nós-Resultado. A Figura 6.6(b) mostra a Tw gerada a partir da consulta Qw.

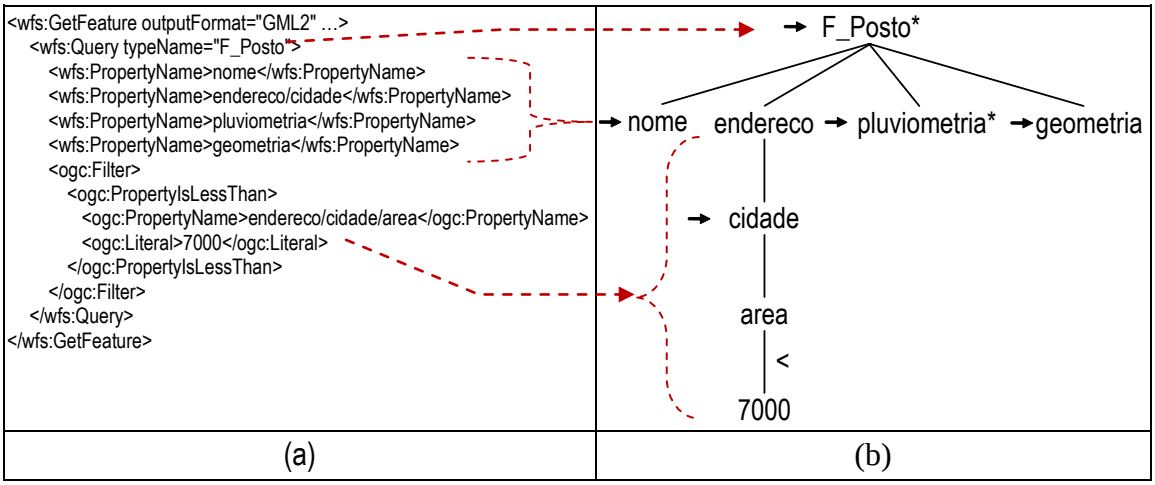


Figura 6.6: (a) Consulta WFS Qw. (b) TPQ Tw.

Passo 2: **Geração da consulta SQL/XML**. Neste passo, o processo é similar ao processo de tradução de consultas XQuery descrito no Capítulo 5. Em síntese, cada caminho de Tw é analisado, e através dos TCs, o caminho é substituído por uma subconsulta SQL/XML correspondente. A Figura 6.7 mostra a consulta SQL/XML Qsw gerada. A consulta está marcada por fragmentos de subconsultas SQL/XML, especificando sua tabela *template* e o caminho de onde a mesma foi retirada dos TCs.

```
SELECT  XMLELEMENT("GML:FEATUREMEMBER",                    TEMPLATE: TF_Posto,  CAMINHO: F_Posto*
XMLELEMENT("F_POSTO",
   XMLFOREST(P.NOME As "nome"),                            TEMPLATE: TF_Posto,  CAMINHO:  nome
   (SELECT XMLELEMENT("CIDADE",                            TEMPLATE: TF_Posto,  CAMINHO:  endereco/cidade
                XMLFOREST(M.NOME AS "NOME"),
                XMLFOREST(M.AREA AS "AREA") )
       FROM DUAL),
   (SELECT XMLAGG(XMLFOREST("PLUVIOMETRIA",                TEMPLATE: TF_Posto,  CAMINHO:  pluviometria
                        XMLFOREST(PL.MES AS "MES"),
                        XMLFOREST(PL.VALOR AS "VALOR")))
    FROM PLUVIOMETRIA_REL PL
    WHERE P.CODPOSTO = PL.CODPOSTO),
    XMLFOREST(SDO_UTIL.TO_GMLGEOMETRY(P.GEOM_POINT)        TEMPLATE: TF_Posto,  CAMINHO:  geometria
                AS "GEOMETRIA")
FROM POSTO_REL P, MUNICÍPIO_REL M
WHERE P.CODMUN = M.CODMUN AND M.AREA < 7000                TraduzPredicado(endereço/cidade/area < 7000)
```

Figura 6.7: Consulta SQL/XML Q_{SW}.

Em seguida, o componente *Query Translator* envia a consulta SQL/XML Qsw ao SGBD. O componente *Query Translator* recebe o resultado da consulta, gera o documento GML e envia para o usuário/aplicações.

Considere o estado do banco de dados Postos (Figura 6.1) apresentado na Figura 6.8 no momento em que a consulta SQL/XML Qsw é enviada ao SGBD. A Figura 6.9 mostra o documento GML resultante.

Posto_rel

cod_posto	nome	rua	cep	fone	fax	geom_point	codorgao	codmun
112	Paramoti	R. São Pedro, 367	59615213	883552365	883552365	-3.1,-39.2333	1	2310407
164	Serragem	R. Prinicpal s/n	60125070	883480168	883480169	-3.45,-38.5	1	2309458
165	Arisco	R. Principal s/n	58021346	883022131	883022132	-3.65,-38.55	2	2310704

Orgao_rel

codorgao	nome	fone	fax
1	FUNCEME	....	.....
2	SUDENE	….	….

Municipio_rel

codmun	nome	area
2309458	OCARA	3290
2312908	SOBRAL	19820
2310407	PARAMOTI	6910
2310704	PENTECOSTE	13940

Pluviometria_rel

codposto	mes	valor
112	02	93.9
112	03	150
164	01	87.8
164	02	171.6
165	01	50.4

Figura 6.8: Um Estado do Banco de Dados Postos.

```
<?xmlversion="1.0"?>
<wfs:FeatureCollection ...>
<gml:boundedBy>
  <gml:Box ...>  <gml:coordinates>
       -3.1,-40.666, -3.85,-39.23</gml:coordinates>
  </gml:Box></gml:boundedBy>
<gml:featureMember>
  <F_Posto">
     <nome>Paramoti</nome>
     <cidade>
        <nome>Paramoti</nome> <area>6910</area>
     </cidade>
     <pluviometria>
        <mes>02</mes> <valor>93.9</valor>
     </pluviometria>
     <pluviometria>
        <mes>03</mes> <valor>150</valor>
     </pluviometria>
     <geometria>
       <gml:Point srsName="EPSG:4326">
         <gml:coordinates ...>-3.1,-39.233
         </gml:coordinates>
```

```
<gml:featureMember>
  <F_Posto>
     <nome>Serragem</nome>
     <cidade>
        <nome>Ocara</nome> <area>3290</area>
     </cidade>
     <pluviometria>
        <mes>01</mes> <valor>87.8</valor>
     </pluviometria>
     <pluviometria>
        <mes>02</mes> <valor>171.6</valor>
     </pluviometria>
     <geometria>
       <gml:Point srsName="EPSG:4326">
         <gml:coordinates ...>-3.45,-38.5
         </gml:coordinates>
       </gml:Point>
     </geometria>
  </F_Posto>
</gml:featureMember>
</wfs:FeatureCollection>
```

```
        </gml:Point>
      </geometria>
    </F_Posto>
</gml:featureMember>
```

Figura 6.9: Documento GML resultante de Q_{SW}.

6.4 Conclusão

Neste Capítulo, mostramos uma adaptação do nosso *framework* à especificação WFS. Para tanto, propomos o serviço *WFService*, o qual provê uma interface para publicação e consulta de visões GML sobre base de dados relacional.

Apresentamos o processo de publicação de visões GML no ***RelP***, o qual é semelhante ao de publicação de uma visão XML, a diferença está somente no tratamento de dados geográficos. Em resumo, o ***RelP*** publica visões GML através dos seguintes passos: (1) definição do esquema da visão GML; (2) definição das Assertivas de Correspondência entre o esquema da visão e o esquema relacional, (3) geração dos Templates de Consulta da visão e (4) registro e armazenamento do esquema e dos TCs da visão via componente *Catalogue*.

Mostramos também o processo de tradução de consultas WFS, o qual é similar ao processo de tradução de consultas XQuery e é realizado em dois passos. (i) Inicialmente, é gerado uma TPQ a partir da consulta WFS através do algoritmo **GeraTPQG**. (ii) Depois, através do algoritmo **GeraSQL/XML**, é gerada a consulta SQL/XML correspondente a partir da TPQ.

Capítulo 7
Trabalhos Relacionados e Avaliação de Desempenho

Neste Capítulo, apresentamos as propostas mais recentes de publicação de dados relacionais na Web. O Capítulo está organizado da seguinte forma. A Seção 7.1 introduz os principais trabalhos relacionados à publicação de dados relacionais na Web. A Seção 7.2 detalha as propostas para publicação e consulta de dados relacionais através de visões XML. A Seção 7.3 detalha as propostas para publicação e consulta de dados relacionais através de visões GML. A Seção 7.4 apresenta uma análise da eficiência do processamento de consultas do ***RelP****, comparando-o com algumas propostas existentes. Finalmente, a Seção 7.5 apresenta a conclusão.*

7.1 Introdução

A crescente popularidade de dados no formato XML e GML, a ampla utilização de bancos de dados relacionais em diversas áreas de aplicação e a necessidade de compartilhamento de dados em diferentes formatos motivaram o surgimento de diversas propostas para publicação e consulta de dados em bases relacionais no formato XML e GML. Na literatura, os principais *middlewares* de publicação XML de dados relacionais são o *XTables* [18], o *SilkRoute* [11], o *Framework de Três Fases* (*Three-phase Framework*) do Clio [7] e o *XML Publisher* [17], e os principais *middlewares* de publicação GML de dados relacionais são o *Deegree WFS* [44] e o *GeoServer* [47].

No *XTables* e *SilkRoute*, uma visão XML é definida como uma consulta XQuery sobre uma visão XML canônica, a qual representa o esquema relacional no formato XML. Esta consulta especifica o esquema da visão e os mapeamentos que descrevem como o esquema da visão está relacionado com a visão canônica. Uma consulta XQuery sobre a visão XML é traduzida em consultas SQL, e os resultados SQL são etiquetados (*tagged*) produzindo um documento XML. Nesses sistemas, o processamento eficiente de consulta não é garantido, pois, dada uma consulta XQuery,

uma ou mais consultas SQL podem ser geradas na tradução. Além disso, tais sistemas não apresentam formalismo de mapeamento; o mapeamento é feito através de consultas XQuery, as quais são difíceis de gerar e manter.

No *Framework de Três Fases* do Clio, uma visão é definida por um conjunto de *tuple-generating dependencies* (tgds) [7 – 10], os quais representam os mapeamentos. O *framework* recebe como entrada os tgds e: (i) para cada tgd, é extraído os dados relevantes da fonte de dados. Então, (ii) fragmentos XML são gerados a partir dos dados relevantes. Finalmente, (iii) os fragmentos XML são mesclados/agrupados para produzir o resultado final. O *framework* é usado para transformação e transporte de dados entre esquemas fonte-alvo. Portanto, não é possível realizar consultas sobre os esquemas.

No *XML Publisher*, sistema proposto anteriormente ao ***RelP***, uma visão é definida por um esquema XML e um conjunto de ACs. Baseado nas ACs da visão, uma *visão de objeto canônica* é criada sobre o banco de dados. Uma consulta XQuery definida sobre o esquema da visão é primeiro traduzida em uma consulta SQL:92 sobre o esquema da visão canônica e através do mecanismo de visão do SGBD, a consulta é executada. A abordagem usada neste sistema é intrusiva, uma vez que a criação da visão de objeto canônica é obrigatória. Além disso, o padrão SQL/XML não foi usado no processo de tradução de consultas.

No *Deegree WFS*, uma visão GML é definida sobre um esquema relacional, e uma consulta WFS sobre essa visão é traduzida em uma ou mais consultas SQL sobre o banco de dados. Os resultados das consultas são etiquetados pelo sistema e integrados para produzir um único documento GML. Esse sistema não garante a eficiência no processamento de consultas, uma vez que, uma consulta WFS pode ser traduzida em mais de uma consulta SQL.

No *GeoServer*, uma visão GML é definida sobre uma única tabela relacional de modo que não é possível navegar pelas chaves estrangeiras para obter dados de outras tabelas. Uma vantagem é que o processo de tradução de consulta no *GeoServer* é simples, uma vez que a tradução de uma consulta WFS em uma consulta consulta SQL é direta.

Além dos *middlewares* e *frameworks*, diversos SGBD foram estendidos para solucionar o problema de publicação de dados relacionais no formato XML [48, 6, 49, 3, 5].

No SGBD *SQL Server* [48], a criação de visões XML utiliza esquemas anotados (*annoted schemas*), os quais são complexos e seguem uma definição não padronizada e proprietária. O *SQL Server* provê a cláusula `FOR XML` para a transformação de resultados de consultas SQL em XML, e também permite consultas utilizando as linguagens **SQLXML**[2] [48] e XQuery.

No SGBD *postgreSQL* [6], os usuários podem consultar e criar visões SQL/XML sobre dados relacionais. Contudo, suporta somente a definição de consultas XPath [41] sobre essas visões.

No SGBD *MySQL* [49], o suporte XML é dado através de aplicações proprietárias, pois ele não possui um tipo de dados XML dedicado. Por exemplo, a ferramenta S*tylus Studio XML Enterprise Suite* [74] permite a definição de consultas XQuery sobre o banco relacional.

Atualmente, os SGBDs *Oracle* [3] e DB2 [5] são os únicos que suportam a criação e consulta de visões XML utilizando o padrão SQL/XML e a linguagem XQuery.

A seguir, na Seção 7.2, apresentamos com mais detalhes o *XTables*, o *SilkRoute*, o *Framework de Três Fases* do Clio e o *Oracle*. Na Seção 7.3, apresentamos o *Deegree WFS* e o *GeoServer*, e na Seção 7.4, apresentamos análises da eficiência do processamento de consultas do ***RelP***.

7.2 Propostas de publicação de dados relacionais no formato XML

Nos últimos anos, XML vem se firmando como padrão para publicação e troca de dados na Web, principalmente devido a sua flexibilidade para representar dados estruturados e semi-estruturados. Contudo, muitas aplicações Web ainda mantêm seus dados armazenados em bancos de dados relacionais. Assim, o problema de publicar dados relacionais no formato XML tem bastante relevância. A seguir, apresentamos em maiores detalhes os principais sistemas de publicação XML de dados relacionais.

[2] SQLXML é uma linguagem de consulta que permite recuperar dados diretamente da base de dados no formato de XML. SQLXML é proprietária da Microsoft Corporation.

7.2.1 XTables

SQLXML é uma linguagem de consulta que permite recuperar dados diretamente da base de dados no formato de XML. SQLXML é proprietária da Microsoft Corporation.
O XTables é um *framework* proposto pela *IBM Almaden Research Center* que permite ao usuário: (i) criar e consultar visões XML sobre um banco relacional; e (ii) armazenar e consultar documentos XML em uma base relacional.

Publicação da Visão

Para publicar visões XML, primeiramente o XTables gera automaticamente uma visão XML canônica do esquema da base relacional sobre o qual será criada a visão. Em seguida, os usuários podem, através de consultas XQuery, definir visões XML sobre a visão XML canônica. No XTables, o estado da visão XML canônica é formado por um elemento raiz, o qual é o nome do banco de dados, e seus descendentes diretos, os quais correspondem às tabelas relacionais do banco. Os elementos relacionados às tabelas possuem subelementos row, correspondendo às linhas da tabela do banco de dados. Cada elemento row possui um subelemento para cada atributo da tabela contendo o valor do referido atributo.

Considere por exemplo, o esquema relacional do banco de dados Empresa apresentado na Figura 7.1 e seu estado apresentado na Figura 7.2. O estado da visão canônica a partir do banco Empresa é apresentado na Figura 7.3(a). A Figura 7.3(b) mostra a visão Departamento criada pelo usuário, que lista todos os departamentos com seus respectivos empregados e endereços. Nessa visão, a cláusula view("empresa") corresponde à visão XML canônica.

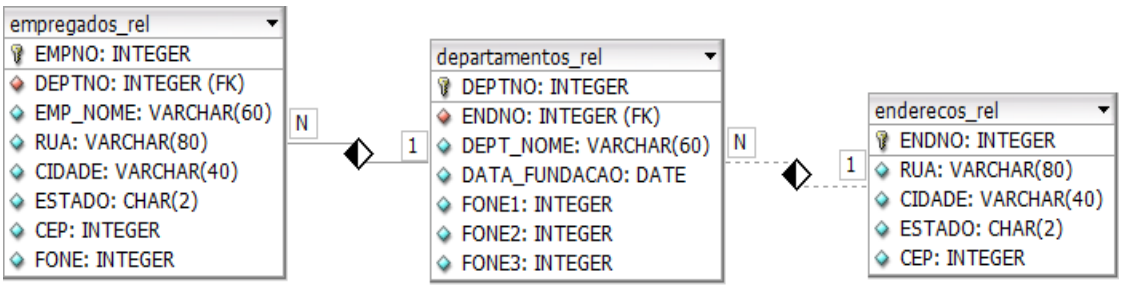


Figura 7.1: Esquema relacional Empresa.

EMPREGADOS_REL

EMPNO	DEPTNO	EMP_NOME	RUA	.....
0010	1010	Smith	....	....
0011	1020	Van	....	....
0012	1010	Mary	....	....

DEPARTAMENTOS_REL

DEPTNO	ENDNO	DEPT_NOME	DATA_FUNDACAO	.....
1010	1	Automação	25/10/2001	....
1020	1	Vendas	10/07/2000	....

Figura 7.2: Um Estado do BD Empresa.

```
<Empresa>
    <empregados_rel>
        <row>
            <empno> 0010 </ empno>
            <deptno> 1010 </ deptno >
            <emp_nome> Smith </emp_nome>
            <rua> ...</rua>
             ...
        </row>
        <row>
            <empno> 0011 </ empno>
            <deptno> 1020</ deptno >
            <emp_nome> Van </emp_nome>
            <rua> ...</rua>
             ...
        </row>
    </ empregados_rel >
    <departamentos_rel>
        <row> ... </row>
        <row> ... </row>
    </ departamentos_rel >
    <enderecos_rel>
        <row> ... </row>
        <row> ... </row>
    </ enderecos _Rel>
</Empresa>
```

(a)

```
create view Departamento as (
    for $dept in view("Empresa")/departamentos_rel/row
    return <departmento id=$dept/deptno>
            <nome> $dept/dept_nome </nome>
            <endereco>
              for $end in view("Empresa")/enderecos_rel/row
              where $dept/endno = $end/endno
              return <rua> $end/rua </rua>
                     <cidade> $end/cidade </cidade>
                     <estado> $end/estado </estado>
                     <cep> $end/cep </cep>
            </endereco>
            <empregados>
              for $emp in view("Empresa")/empregados_rel/row
              where $dept/deptno = $emp/deptno
              return <empregado>
                       <nome> $emp/emp_nome </nome>
                       <rua> $emp/rua </rua>
                     </empregado>
            </empregados>
            </departmento>
)
```

(b)

Figura 7.3: (a) Estado da visão canônica do banco Empresa. (b) Visão Departamento.

Tradução de Consulta

No XTables, a tradução de uma consulta XQuery sobre uma visão é realizada nos seguintes passos:

Passo 1: **geração da *XML Query Graph Model* (XQGM)** [18]. Primeiramente, uma consulta XQuery é reescrita em uma consulta XQGM. XQGM é um modelo de representação intermediário que facilita a manipulação do plano de consulta no processo de tradução. XQGM apresenta um conjunto de operadores e funções que capturam a semântica da consulta XQuery. Tais operadores são subconjuntos dos operadores relacionais tradicionais, como *project*, *select*, *join*, etc.

Passo 2: geração das consultas SQL equivalentes. Com base na consulta XQGM e nas regras de mapeamento da visão canônica com o banco relacional, são geradas uma ou mais consultas SQL. Para tanto, a XQGM inicial em termos da visão XML é reescrita em termos da visão canônica (fase de expansão e otimização). Em seguida, a(s) consulta(s) SQL é (são) enviada(s) ao banco de dados.

Passo 3: geração do resultado XML. Com base na XQGM, os resultados das consultas SQL são formatados, gerando um único documento XML que é retornado para o usuário.

Considere, por exemplo, a consulta XQuery Q_X da Figura 7.4(a), a qual lista os empregados que trabalham no departamento de Automação. XTables gera, a partir de Q_X, a XQGM apresentada na Figura 7.4(b). Cada valor correspondente de cada cláusula da consulta XQuery é traduzida em uma operação XQGM. Por exemplo, o operador Select computa o predicado da consulta.

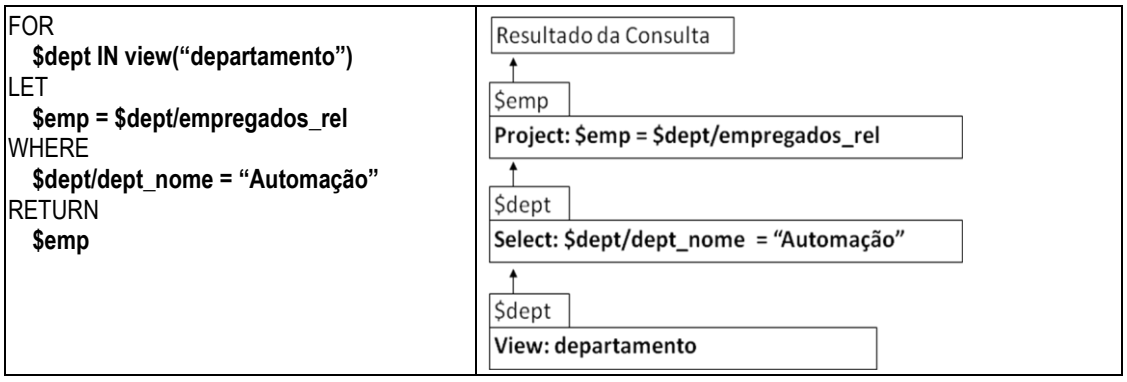
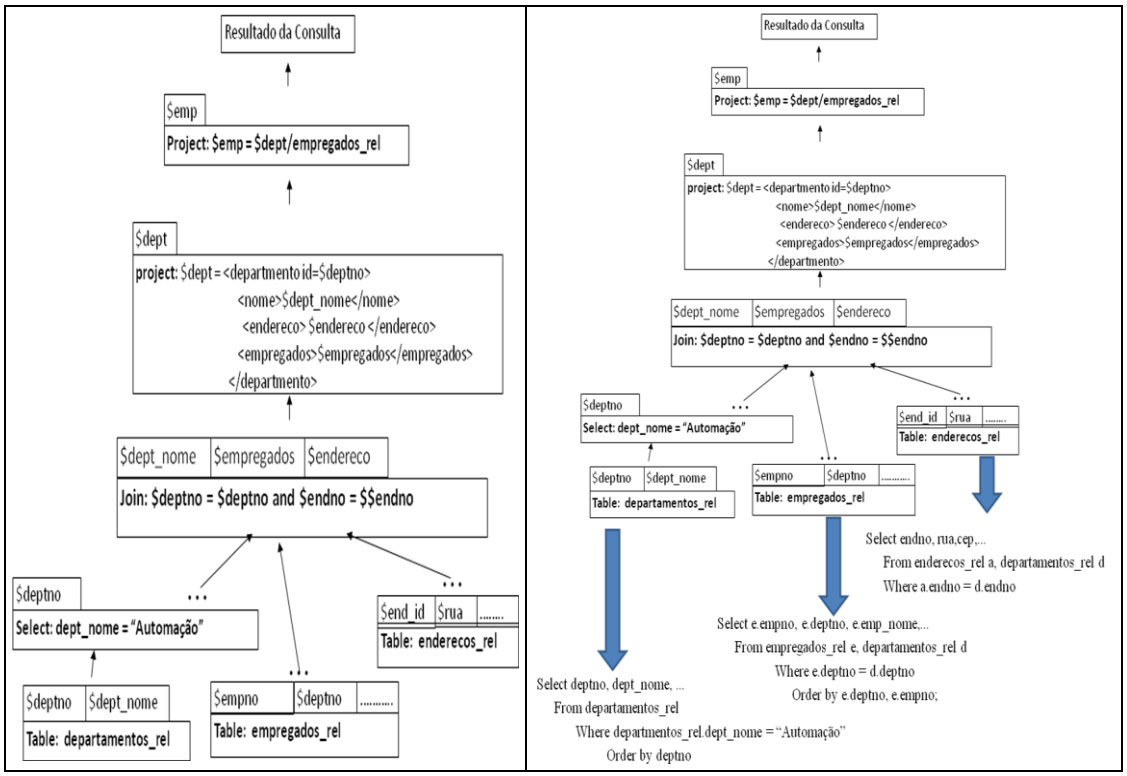


Figura 7.4: (a) Consulta XQuery Q_X. (b) XQGM obtida da consulta Q_X.

Em seguida, é executada a fase de expansão e otimização da XQGM, a qual consiste em substituir as operações no nível de visão (da XQGM inicial) em operações no nível de tabela. As mesmas regras de otimização de consultas utilizadas em banco de dados tradicionais são aplicadas. A Figura 7.5(a) apresenta a consulta XQGM gerada após a fase de expansão e otimização sobre a consulta da Figura 7.4.

Após a fase de expansão e otimização, a XQGM é traduzida em uma ou mais consultas SQL. Para gerar a(s) consulta(s) SQL, o XTables traduz todas as operações e funções da XQGM em uma ou mais consultas SQL. Se mais de uma consulta SQL é formada, o XTables as executa ordenadamente no banco de dados. A Figura 7.5(b) mostra as consultas SQL geradas a partir da XQGM otimizada.

Por fim, a XQGM é novamente utilizada para a geração do resultado XML, o qual é enviado para o usuário. A Figura 7.6 mostra o documento XML de saída.



(a) (b)

Figura 7.5: (a) XQGM expandida e otimizada. (b) Processo de geração das consultas SQL.

```
<empregados>
        <empregado>
                <nome> Smith <nome>
                <rua> ... </rua>
        </empregado>
</ empregados>
```

Figura 7.6: Documento XML de saída.

7.2.2 SilkRoute

O SilkRoute é um *framework* proposto pela *AT&T Labs Research*, que, assim como o XTables, permite ao usuário criar e consultar visões XML. No SilkRoute, o processo de publicação de visões é similar ao XTables, ou seja, o usuário cria visões XML através de consultas XQuery sobre uma visão XML canônica do banco de dados. Desde modo, iremos apenas apresentar o processo de tradução de consultas.

Tradução de Consulta

No SilkRoute, a tradução de uma consulta XQuery sobre uma visão é realizada nos seguintes passos:

Passo 1: Geração da consulta composta. Primeiramente, uma *consulta composta* é gerada a partir da combinação da consulta XQuery do usuário com a consulta XQuery que define a visão. Em resumo, a consulta XQuery definida sobre a visão XML é reescrita em uma consulta XQuery (consulta composta) definida sobre a visão canônica. A consulta composta é modelada usando uma *view forest*, a qual é uma árvore que representa a estrutura do resultado da consulta composta juntamente com as cláusulas necessárias para computá-la.

Passo 2: Geração das consultas SQL equivalentes. Com base na *view forest* da consulta composta e nas regras de mapeamentos da visão canônica com o banco relacional, são geradas uma ou mais consultas SQL. Em resumo, o SilkRoute percorre no sentido *bottom-up* a *view forest* para formar a(s) consulta(s) SQL. Em seguida, a(s) consulta(s) SQL é (são) enviada(s) ao banco de dados. Se mais de uma consulta SQL for gerada, então estas serão unidas através das operações *union* ou *join*.

Passo 3: Geração do resultado XML. Com base na *view forest* da consulta composta, o SilkRoute adiciona as *tags* XML ao resultado das consultas SQL, gerando o documento XML de saída.

Considere por exemplo, o esquema relacional do banco de dados Empresa apresentado na Figura 7.1 e seu estado apresentado na Figura 7.2. Considere, a visão XML Departamento criada pelo usuário apresentada na Figura 7.3(b) e a consulta XQuery Qx (Figura 7.4(a)), definida sobre a visão Departamento. SilkRoute gera, a partir de Q_X, a consulta composta apresentada na Figura 7.7. Em síntese, a consulta composta é gerada substituindo as expressões de caminho sobre a visão XML pelas expressões de caminho/subconsultas equivalentes sobre a visão canônica. Por exemplo, a expressão $emp da cláusula RETURN de Qx é substituída pela subconsulta <empregados> for $emp in ...</empregados> da visão XML, pois essa subconsulta computa os empregados da visão a partir da visão canônica.

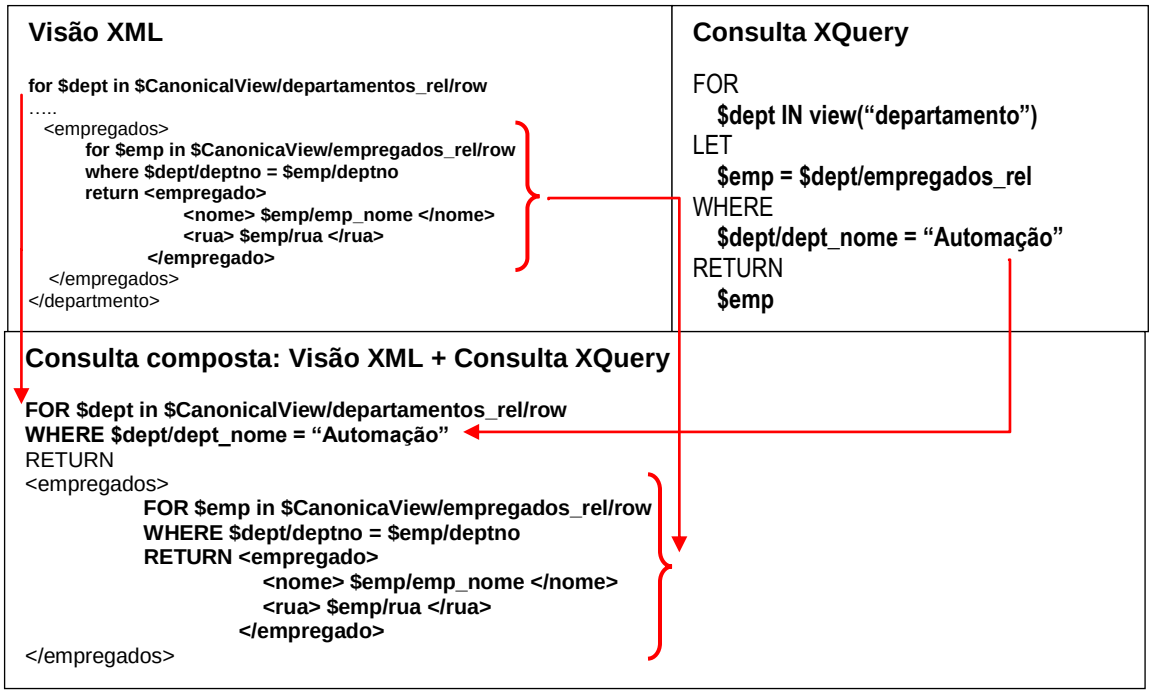
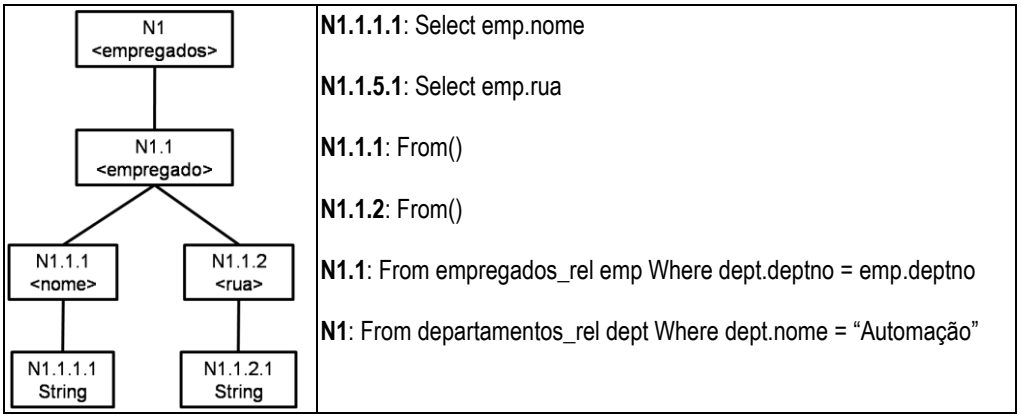


Figura 7.7: *Consulta composta*: Visão XML e Consulta XQuery.

A *view forest* da consulta composta pode ser vista na Figura 7.8(a), na qual seus nós correspondem aos elementos XML da cláusula RETURN da consulta. Cada nó da *view forest* possui uma subconsulta SQL, a qual corresponde ao mapeamento para o banco relacional como mostra a Figura 7.8(b). As subconsultas SQL são geradas analisando as expressões de caminho e os valores das cláusulas da consulta composta. Por exemplo, o nó raiz N1 captura o valor correspondente à tabela relacional definida nas cláusulas FOR e WHERE da consulta composta, anteriores e mais próximas ao elemento <empregados>.



(a) (b)

Figura 7.8: (a) *view forest* da consulta composta. (b) Valores dos nós da *view forest*.

Em seguida, o SilkRoute percorre no sentido *bottom-up* a *view forest* para gerar a consulta SQL. A Figura 7.9 mostra a consulta SQL gerada a partir da *view forest* apresentada na Figura 7.8.

```
Select emp.name, emp.rua
From empregados_rel emp, departamentos_rel dept
Where dept.deptno = emp.deptno and dept.nome = “Automação”
Order by emp.nome, emp.rua;
```

Figura 7.9: Consulta SQL resultante.

Por fim, a consulta SQL é executada no banco de dados e o SilkRoute adiciona as *tags* XML ao resultado das consultas SQL, gerando o documento XML de saída. O documento XML de saída gerado a partir da consulta apresentada na Figura 7.9, é apresentado na Figura 7.6.

7.2.3 *Framework* de Três Fases do Clio

O Clio [7 – 10] é uma ferramenta semi-automática para a criação de mapeamentos entre esquemas, desenvolvida pela *IBM Almaden Research Center*. Clio apresenta uma interface amigável para definir mapeamentos declarativos entre esquemas e gerenciar tarefas complexas de transformação e integração de dados heterogêneos, tais como XML e dados relacionais.

O Clio permite ao usuário fazer mapeamentos entre esquemas de bancos de dados relacionais, documentos XML e bancos de dados XML. Os esquemas fonte-alvo podem ser: Relacional – Relacional, Relacional – XML, XML – Relacional e XML – XML.

De acordo com a arquitetura do Clio apresentada na Figura 7.10, o processo de transformação de dados do esquema fonte em dados do esquema alvo consiste de três passos. (i) As correspondências entre os esquemas fonte e alvo são geradas pelo usuário através de uma interface gráfica ou através do componente s*chema matching*. Em seguida, (ii) o componente *mapping generation* analisa as correspondências entre os esquemas e gera o *schema mapping*, o qual consiste de um conjunto de mapeamentos lógicos, também chamados de *tuple-generating dependencies* (tgd). Tgds são assertivas

declarativas, expressas como restrições entre esquemas fonte e alvo. Por fim, (iii) o componente *query generation* converte o conjunto de mapeamentos lógicos em um *script* de transformação executável. Esse *script* é uma consulta (SQL, XQuery, XSLT,...) que fará a transformação dos dados físicos do esquema fonte nos dados físicos do esquema alvo.

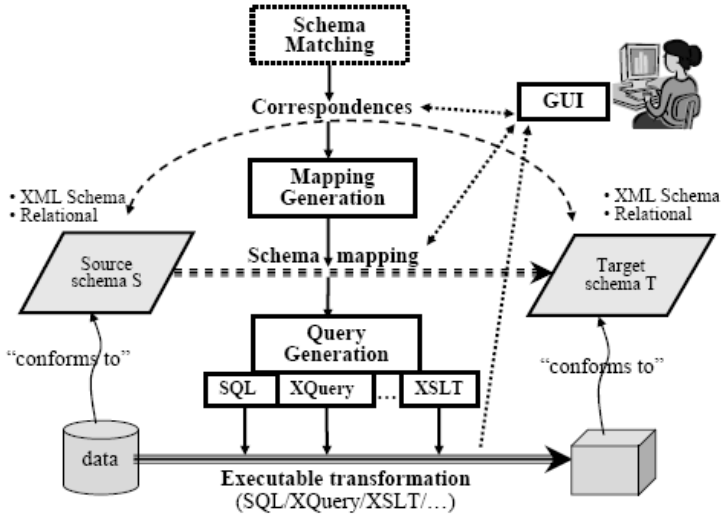


Figura 7.10: Arquitetura do sistema Clio.[3]

Em 2007, o Clio apresentou um *framework* denominado *Framework de Três Fases* (*Three-phase Framework*) que faz transformações: Relacional – XML e XML – XML. Como apresentado na Figura 7.11, o Framework de Três Fases recebe como entrada o *schema mapping* e (i) para cada tgd, extrai da fonte os dados relevantes para o tgd, depois (ii) gera fragmentos XML a partir dos dados, e por fim, (iii) realiza a mesclagem/agrupamento dos fragmentos XML.

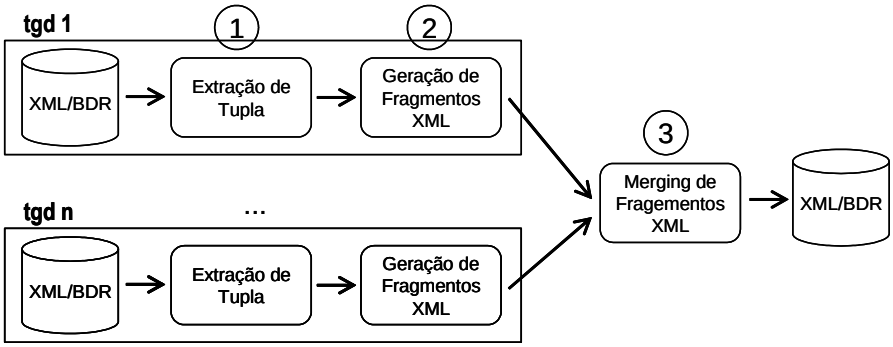


Figura 7.11: Modelo de execução de Três Fases para Transformação dos Dados.[4]

[3] Hass [9] (2005)
[4] Jiang [7] (2007)

Considere o esquema relacional apresentado na Figura 7.12(a) e o esquema XML apresentado na Figura 7.12(b). O símbolo '*' corresponde a um conjunto de valores e as setas no esquema relacional correspondem aos relacionamentos entre tabelas. A Figura 7.13 mostra as correspondências entre o esquema relacional e XML realizadas pelo usuário.

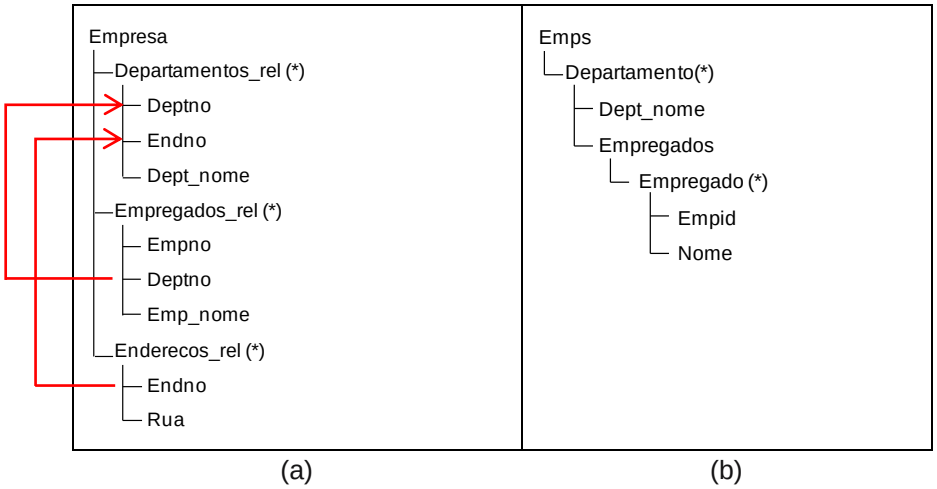


Figura 7.12: (a) Esquema Relacional Fonte. (b) Esquema XML Alvo.

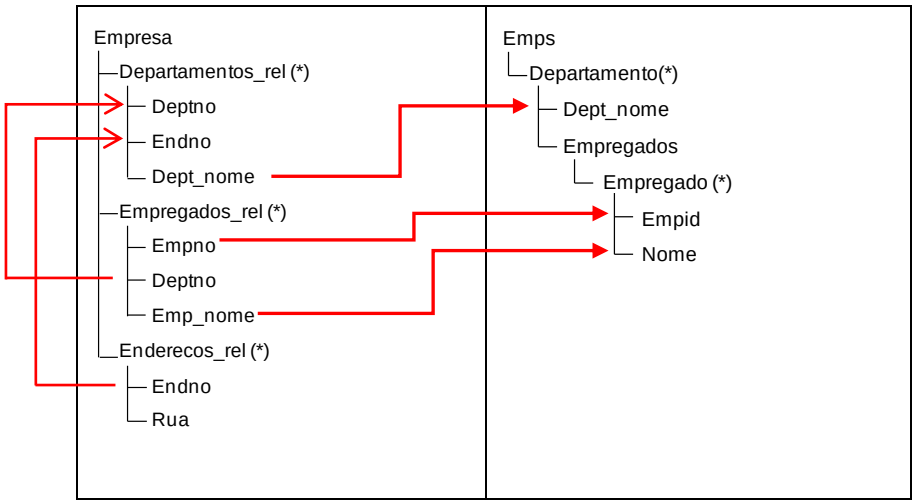


Figura 7.13: Correspondências entre esquema Relacional e XML da Figura 7.12.

O componente *mapping generation* captura as correspondências e gera o(s) tgd(s). Um tgd pode ser constituído de uma cláusula *FOR*, *EXISTS* e *WHERE*. No caso, a cláusula *FOR* é formada pelas tabelas e uma possível junção entre elas, a cláusula *EXISTS* é formada por elementos do esquema XML e a cláusula *WHERE* faz junções entre tabelas da cláusula *FOR* com elementos da cláusula *EXISTS*. Por exemplo, o Tgd 1 e o Tgd 2 apresentado na Figura 7.14 foram gerados a partir das correspondências da

Figura 7.13. O tgd 1 pode ser lido como: para as tabelas mapeadas (cláusula *FOR*), onde (cláusula *WHERE*) existe um relacionamento entre essas tabelas, existe (cláusula *EXIST*) elementos multivalorados XML, onde (cláusula *WHERE*) o mapeamento entre os elementos dos esquemas são descritos através de junções. O tgd 2 transforma os dados de departamento_rel sem qualquer empregado e pode ser lido similarmente como o tgd 1. Após a geração dos tgds, o Framework de Três Fases faz a transformação dos dados como apresentado nas fases a seguir.

```
Tgd 1:  FOR d in db.departamentos_rel, de in db.empregados_rel
                WHERE d.deptno = de.deptno
        EXISTS d' in Emps.departamento, de' in d'empregados.empregado
        WHERE de'.empid = de.empno
                and de'.nome = de.emp_nome
                and d'.dept_nome = d.dept_nome

Tgd 2:  FOR d in db.departamentos_rel
        EXISTS d' in Emps.departamento
        WHERE d'.dept_nome = d.dept_nome
```

Figura 7.14: Mapeamentos Lógicos ou tgds.

Fase 1 - ***Tuple Extraction*:** Esta fase corresponde à extração de tuplas. Para extrair as tuplas do esquema relacional, consultas SQL serão geradas a partir dos mapeamentos lógicos. A Figura 7.15 mostra as consultas SQL geradas a partir dos tgds da Figura 7.14. Se mais de uma consulta SQL for gerada, então elas são combinadas pela operação *left-inner-join*. No exemplo, as consultas SQL 1 e SQL 2 foram combinadas, executadas e o resultado da consulta é apresentado na Figura 7.16.

```
SQL 1: SELECT d.dept_nome, emp.empno, emp. emp_nome
       FROM departamentos_rel d, empregados_rel emp
       WHERE d.deptno = emp.deptno

SQL 2: SELECT d.dept_nome
       FROM departamentos_rel d
```

Figura 7.15: Consultas SQL gerada a partir dos tgds da Figura 7.14.

DEPT_NOME	EMPNO	EMP_NOME
Automação	10	Smith
Automação	11	Cock
Vendas	9	Van

Figura 7.16: Dados extraídos da primeira Fase.

Fase 2 - *XML-fragment generation*: Nesta fase, cada tupla é transformada em um fragmento XML. Para isso, com base nos tgds, é gerado um *template* que constrói os elementos da visão a partir das tuplas relacionais obtidas da Fase 1. A Figura 7.17 mostra o *template* e a primeira tupla sendo gerada.

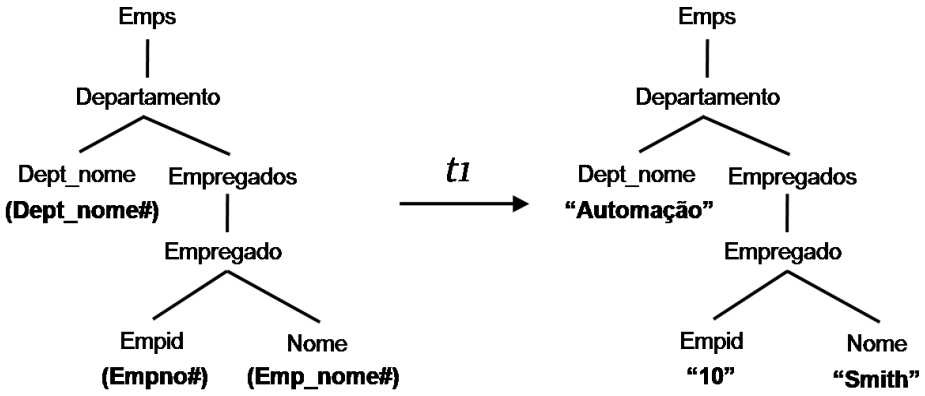


Figura 7.17: *Template* e Fragmento da tupla t1 da tabela.

Fase 3 - *XML-fragment merging/grouping*: Nesta fase, é feita a mesclagem e/ou agrupamento dos fragmentos XML gerados na 2ª Fase. O resultado é um documento XML não redundante e coerente com o esquema alvo. A Figura 7.18 mostra a árvore do documento XML de saída.

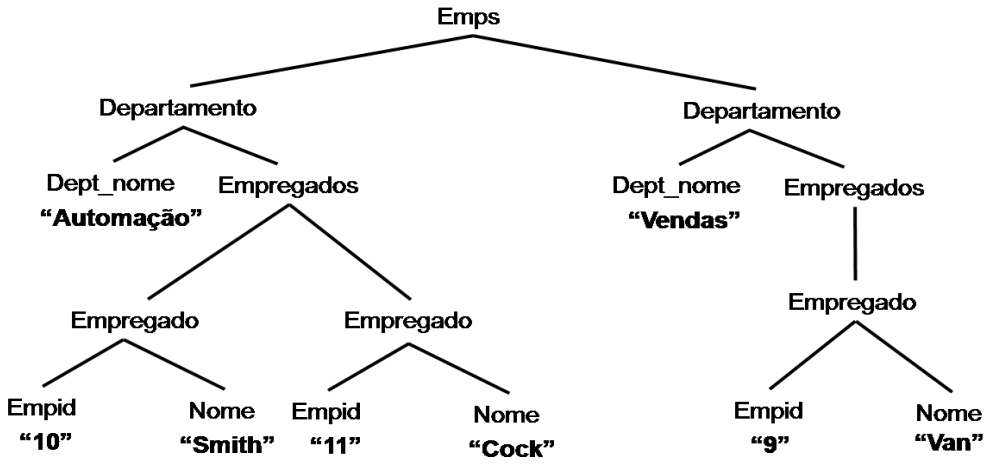


Figura 7.18: Árvore do documento XML resultante.

Apesar do alto-nível da linguagem de mapeamento, o *framework* apresentado é usado somente na publicação dos dados do esquema fonte no formato do esquema da visão XML e não permite a definição de consultas sobre a visão.

7.2.4 Oracle 10g

Oracle XML DB é a tecnologia existente no Oracle 10g que oferece armazenamento e manipulação de documentos XML [24]. Esta tecnologia estende o banco de dados relacional Oracle, oferecendo muitas das funcionalidades associadas a bancos de dados XML nativos.

O XMLType é um tipo de objeto nativo do Oracle que permite que o banco de dados entenda que uma coluna ou uma tabela contém documentos XML. O XMLType também provê métodos que permitem operações comuns realizadas no conteúdo XML, como validação de esquema e transformações XSLT [24, 25]. O tipo XMLType pode ser usado como qualquer outro tipo de dados comum, como, por exemplo, na criação de uma coluna em uma tabela relacional, na declaração de variáveis PL/SQL, ou como retorno de funções (*functions*) em PL/SQL.

O Oracle 10g suporta o padrão SQL/XML e suas funções XMLQuery, XMLTable e XMLExists [25]. A função XMLQuery permite realizar consultas XQuery sobre instâncias do tipo XMLType. Uma consulta XQuery enviada ao banco através desta função é traduzida em uma consulta SQL/XML equivalente. A consulta SQL/XML é executada e o resultado é retornado ao usuário. O processo de reescrita de uma consulta XQuery em uma consulta SQL/XML no Oracle é descrita em [24, 25].

A tradução de consultas XQuery em consultas SQL/XML permite utilizar os mecanismos de otimização já implementados para a execução eficiente de consultas SQL/XML. Os mecanismos de otimização desenvolvidos para o SQL/XML são baseados em otimizadores relacionais clássicos, o que representa um ganho enorme em performance. Desse modo, é possível utilizar a tecnologia objeto-relacional e a infra-estrutura SQL/XML dentro do Oracle para suportar execução nativa de XQuery.

O Oracle também estende o mecanismo de Visão para suportar visões XML. Essas visões são compostas de instâncias do tipo XMLType. Estas instâncias podem existir fisicamente no banco de dados (em tabelas XML) ou podem ser “sintetizadas” a

partir de dados no formato relacional ou de objetos através das funções de publicação do SQL/XML.

7.3 Propostas de publicação de dados relacionais no formato GML

Na área geográfica, a publicação de dados relacionais no formato GML é importante, pois o compartilhamento de informações padronizadas é indispensável devido ao alto custo de produção de dados espaciais e grande complexidade de análises ambientais que os envolvem. Desta forma, apresentamos duas importantes propostas para publicação GML de dados relacionais: o Deegree WFS e o GeoServer.

7.3.1 Deegree WFS

Deegree é um *framework* que oferece um conjunto de serviços Web para infra-estrutura de dados espaciais. Sua arquitetura foi desenvolvida usando padrões do *Open Geospatial Consortium* (OGC) e da ISO/TC 211. O Deegree é um software livre de licença GNU LGPL (*GNU Lesser General Public License*) que foi desenvolvido no Departamento de Geografia da Universidade de Bonn, na Alemanha.

Um dos serviços oferecidos pelo Deegree é o Deegree WFS, o qual provê ao usuário um servidor WFS para acesso e manipulação de fontes de dados relacionais através de visões GML. O Deegree WFS foi desenvolvido na linguagem Java e oferece suporte aos SGBDs PostgreSQL/PostGIS, SQL Server, Oracle e MySQL; e também dá suporte a *shapefiles*.

Publicação da Visão

O Deegree WFS publica visões GML através da configuração de dois arquivos XML: wfs_configuration e datastore configuration. O arquivo wfs_configuration descreve as capacidades do servidor e é dividido em diversas seções, dentre elas,

destacamos a seção <FeatureTypeList>, a qual descreve a lista de visões GML que o servidor disponibiliza. O arquivo datastore configuration especifica o mapeamento entre o esquema da visão GML e o esquema do banco de dados relacional.

Para publicar visões GML, primeiramente o usuário define a visão GML no arquivo wfs_configuration, no qual o usuário deve adicionar um elemento FeatureType na seção <FeatureTypeList>, seguido dos subelementos: name, title, abstract, srs e LatLonBoundingBox. Tais elementos identificam a visão GML no servidor. Em seguida, no arquivo datastore configuration o usuário define o esquema da visão GML, constrói os mapeamentos com o esquema relacional e define os dados de conexão com o banco de dados. O esquema da visão e os mapeamentos devem ser construídos seguindo o formalismo dado no Deegree WFS.

Considere, por exemplo, o esquema da visão GML CidAdm apresentada na Figura 7.19(a). CidAdm foi publicada a partir do esquema relacional Cidades, apresentado na Figura 7.19(b). A Figura 7.20 mostra a seção <FeatureTypeList> do arquivo wfs_configuration. O elemento <FeatureType> define a visão GML CidAdm.

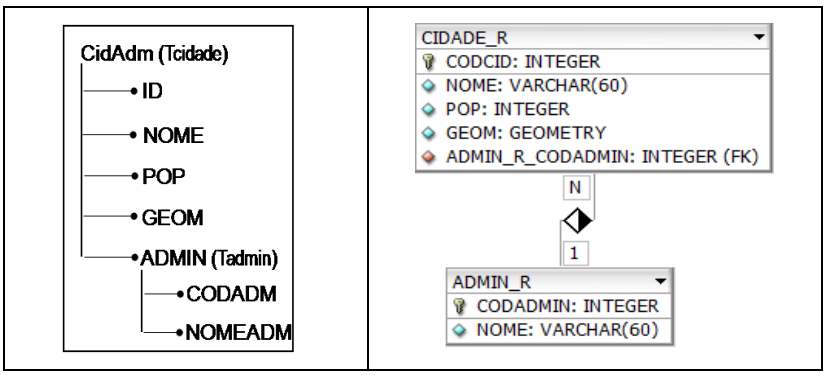


Figura 7.19: (a) Esquema da visão CidAdm. (b) Esquema relacional Cidades.

```
<?xml version="1.0" encoding="UTF-8"?>
<WFS_Capabilities version="1.0.0">
…
<FeatureTypeList>
   <Operations>
      <Query/>
   </Operations>
   <FeatureType>
      <ResponsibleClass
         className="org.deegree_impl.services.wfs.oracle.OracleDataStore"
         configURL="file:///C:/deegreewfs/WEB-INF/xml/cidades.xml"/>
      <Name>CidAdm</Name>
      <Title>A cidade e seu administrador </Title>
      <Abstract>Descreve dados da cidade e quem é seu administrador</Abstract>
      <SRS>EPSG:4326</SRS>
      <LatLonBoundingBox minx="0" miny="0" maxx="180" maxy="90"/>
   </FeatureType>
   . . .
</FeatureTypeList>
</WFS_Capabilities>
```

Figura 7.20: Arquivo *wfs_configuration* do Deegree WFS.

A Figura 7.21 apresenta parte do arquivo datastore configuration da visão CidAdm. Dentre seus elementos XML, podemos destacar:

i. <xsd:annotation> e <xsd:appinfo>: esses elementos apresentam os parâmetros de conexão do banco de dados e o mapeamento entre os elementos do esquema relacional com o esquema da visão;

ii. <MappingField>: define o relacionamento entre os elementos da visão GML com os atributos das tabelas. Por exemplo, nas linhas 12–16, o elemento ID no esquema da visão corresponde ao atributo CODCID da tabela CIDADE_R;

iii. <Relation>: define na visão GML o relacionamento entre as tabelas do banco de dados. Por exemplo, nas linhas 32–41, é definida a relação entre a tabela CIDADE_R e a tabela ADMIN_R.

```
1.  <xsd:schema <!– namespaces -> ><!– configuração de conexão com o banco de dados ->
2.  <xsd:annotation>
3.      <xsd:appinfo>  <!- - declarações de conexão com o banco de dados ->   </xsd:appinfo>
4.  </xsd:annotation>
5.  <xsd:element name= "CidAdm" type="app:CidadeType" substitutionGroup="gml:_Feature">   <xsd:annotation>  6.
6.     <xsd:appinfo>
7.           <deegreewfs:table>CIDADE_R</deegreewfs:table>
8.           <deegreewfs:visible>true</deegreewfs:visible>
9.           <deegreewfs:transaction update="true" delete="true" insert="true"/>
10.       </xsd:appinfo>   </xsd:annotation>   </xsd:element>     <xsd:complexType name="CidadeType">
11.   ……..
12.   <xsd:element name= "ID" type="xsd:integer">        <xsd:annotation>           <xsd:appinfo>
13.                       <deegreewfs:Content>
14.                                 <deegreewfs:MappingField field= "codcid" type="INTEGER"/>
15.                       </deegreewfs:Content>
16.         </xsd:appinfo>        </xsd:annotation> .       </xsd:element>
17.    ……..
18.   <xsd:element name="GEOM" type="gml:GeometryPropertyType">
19.               <xsd:annotation>
20.           <xsd:appinfo>
21.                       <deegreewfs:Content>
22.                                 <deegreewfs:MappingField field="geom" type="GEOMETRY"/>
23.                       </deegreewfs:Content>
24.           </xsd:appinfo>
25.      </xsd:annotation>
26.   </xsd:element>
27.   <xsd:element name="CODADM" type="xsd:string">
28.      <xsd:annotation>
29.           <xsd:appinfo>
30.                 <deegreewfs:Content>
31.                             <deegreewfs:MappingField field="codadmin" type="VARCHAR"/>
32.                          <deegreewfs:Relation>
33.                               <deegreewfs:From>
34.                                     <deegreewfs:MappingField field="codpref" type="INTEGER"/>
35.                               </deegreewfs:From>
36.                               <deegreewfs:To fk="true">
37.                                  <deegreewfs:MappingField field="codadmin" type="INTEGER"
38.                                                          table="ADMIN_R"/>
39.                               </deegreewfs:To>
40.
41.   </deegreewfs:Relation></deegreewfs:Content></xsd:appinfo></xsd:annotation></xsd:element>
```

Figura 7.21: Arquivo *datastore* da visão GML *CidAdm*.

Tradução de Consulta

No Deegree WFS, o processo de tradução de consultas consiste em reescrever uma consulta WFS em uma ou mais consultas SQL. A tradução de uma consulta WFS sobre uma visão GML é realizada em duas fases: (i) através do arquivo de mapeamento datastore configuration, uma consulta WFS é traduzida em uma ou mais consultas SQL semanticamente equivalentes, e, (ii) a partir do resultados SQL, é gerado um documento GML resultado que é enviado ao usuário.

Considere a visão GML CidAdm apresentada na Figura 7.19(a) e a consulta WFS Qw da Figura 7.22, a qual retorna o nome, população e geometria das cidades com mais de 5 mil habitantes.

```
<?xml version="1.0" ?>
<wfs:GetFeature service="WFS"
  <!-- definição de namespaces -->
  <wfs:Query TypeName="CidAdm">
          <ogc:PropertyName>NOME</ogc:PropertyName>
          <ogc:PropertyName>POP</ogc:PropertyName>
          <ogc:PropertyName>GEOM</ogc:PropertyName>
          <ogc:Filter>
              <ogc:PropertyIsGreaterThan>
                  <ogc:PropertyName>POP</ogc:PropertyName>
                  <ogc:Literal>5000</ogc:Literal>
              </ogc:PropertyIsGreaterThan>
          </ogc:Filter>
  </wfs:Query>
```

Figura 7.22: Consulta WFS Qw.

Para traduzir a consulta Qw, o Deegree WFS obtém a tabela e os atributos relacionais do arquivo datastore configuration a partir dos valores do atributo TypeName, dos elementos PropertyName e do elemento filter de Qw. De acordo com a Figura 7.22, a tabela a ser realizada a consulta é CIDADE_R, os valores Nome, Pop e Geom fazem parte do resultado da consulta e os valores PropertyIsGreaterThan, POP e 5000 formam o predicado da consulta. A Figura 7.23 mostra a consulta SQL gerada a partir de Qw.

Por fim, o resultado da consulta SQL é compilado em um único documento GML que é enviado ao usuário.

```
SELECT W1.nome, W1.pop, W1.geom
FROM CIDADE_R W1
WHERE W1.pop > 5000
```

Figura 7.23: Consulta SQL obtida a partir de Qw.

No Deegree WFS, a configuração manual do servidor obriga a existência de um especialista em XML e nas especificações do OGC. Em geral, configurar arquivos no Deegree WFS é um trabalho de longa duração e complexo, mesmo com a existência de um especialista. Além disso, uma modificação no esquema do banco pode afetar várias visões GML, requerendo que o arquivo de mapeamento, *datastore configuration*, seja redefinido para todas as visões afetadas.

Assim, como o Silkroute e o XTables, o Deegree WFS não garante a eficiência do processamento de consulta, pois dada uma consulta WFS, uma ou mais consultas SQL podem ser geradas na tradução.

7.3.2 GeoServer

O GeoServer é um projeto desenvolvido pela *The Open Planning Project* (TOPP) com parceria da *Refractions Research*, cuja finalidade é fornecer serviços WFS e *Web Map Service* (WMS) seguindo a especificação do OGC. Assim como no Deegree WFS, o GeoServer disponibiliza uma interface permitindo que usuários/aplicações visualizem e manipulem dados geográficos armazenados em fontes de dados através de visões de GML.

O projeto GeoServer foi desenvolvido na linguagem Java, na plataforma J2EE, e dá suporte a uma variedade de banco de dados: PostgreSQL/PostGIS, Oracle, DB2 e MySQL; e também dá suporte a *shapefiles*.

Publicação da Visão

No GeoServer, o processo de publicação de uma visão GML é realizado pelos seguintes passos: (1) o usuário informa os parâmetros de acesso ao banco de dados e o esquema relacional é obtido. Em seguida, (2) o usuário define o esquema da visão escolhendo quais atributos da tabela relacional tornarão elementos da visão. Por fim, (3) o esquema e os mapeamentos são registrados e armazenados em um arquivo codificado, e depois, o arquivo Catalog do GeoServer é atualizado com as informações

da nova visão. O arquivo Catalog define principalmente as visões GML publicadas no servidor.

No GeoServer, o processo de publicação de uma visão GML é realizado através de interfaces gráfica, inclusive a configuração do arquivo Catalog.

Considere, por exemplo, o esquema da visão GML CidAdm2 apresentado na Figura 7.24. CidAdm2 foi publicada a partir do esquema relacional Cidades, apresentado na Figura 7.19(b). A Figura 7.25 mostra parte da seção <datastores> do arquivo Catalog. O elemento <datastore> define uma visão GML.

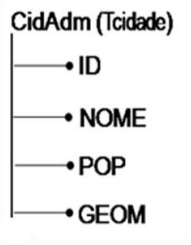


Figura 7.24: Esquema da visão GML CidAdm2.

```
<catalog >
 <datastores >
  <datastore id = "CidAdm" enabled = "true" namespace = "cid" >
   <connectionParams >
        <parameter name = "url" value = "file:data/CidAdm" />
   </connectionParams>
  </datastore>
  …
 </datastores>
….
</catalog>
```

Figura 7.25: Arquivo *catalog* do GeoServer.

Tradução de Consulta

No GeoServer, o processo de tradução de consultas é similar ao Deegree WFS, ou seja, a tradução de consultas WFS sobre uma visão GML é realizada em duas fases: (i) a consulta WFS é traduzida em uma consulta SQL semanticamente equivalente; (ii) e a partir do resultado SQL, é gerado o documento GML correspondente.

No Geoserver, uma visão só pode estar relacionada com uma única tabela do banco de dados. Conseqüentemente, o processo de tradução de consulta é direto: uma consulta WFS é traduzida em uma única consulta SQL.

Apesar das limitações, o uso de ferramentas administrativas que facilitam a publicação de visões GML torna o processo de criação de visões no GeoServer mais fácil que no Deegree WFS.

7.4 Avaliação de desempenho do *RelP*

Nesta seção, apresentamos os experimentos realizados para validação do *framework* proposto nesta dissertação. Os testes têm como objetivo avaliar a eficiência da tradução de consultas dos Sistemas: ***RelP***, Oracle 10g, Deegree WFS v. 2.2.1 e GeoServer v. 1.7.4. Para tanto, realizamos duas diferentes análises: (***A1***) avaliação comparativa do processamento de consultas XQuery do ***RelP*** e do Oracle 10g, e (***A2***) avaliação comparativa do processamento de consultas WFS do ***RelP***, do Deegree WFS e do GeoServer.

Na análise ***A1***, o SGBD Oracle 10g foi escolhido por ser um dos poucos a oferecer um mecanismo de visão que permite a criação de visões SQL/XML e a consulta dessas visões através da linguagem XQuery. Dentre os outros trabalhos relacionados, não foi possível realizar testes com os sistemas XTables e SilkRoute, pois estes projetos foram descontinuados. Além disso, bibliotecas de instalação desses sistemas não estão mais disponíveis na Web e muitos documentos encontram-se inacessíveis.

Na análise ***A2***, o Deegree WFS e o GeoServer foram escolhidos por serem os softwares livres bastante utilizados.

Para as duas análises, os sistemas foram instalados em uma máquina com processador Intel Dual Core 1.79 Ghz com 1 GB RAM, sistema operacional *Windows XP - Service Pack 2*.

Nos experimentos, foram criados dois bancos de dados no Oracle 10g, sendo um para cada análise. Para a análise ***A1***, o banco de dados relacional Empresa, apresentado na Figura 7.1, foi criado com aproximadamente 250 MB de dados. Para a

análise ***A2***, o banco relacional Postos, apresentado na Figura 6.1, foi criado com aproximadamente 250 MB de dados. A Tabela 7.1 resume as informações descritas acima.

	RelP	Oracle 10g	Deegree WFS	GeoServer
Hardware	Intel Dual Core 1.79 Ghz 1 GB RAM			
Sistema Operacional	Windows XP - Service Pack 2			
SGBD Oracle com 250 MB	BD Empresa / BD Postos	BD Empresa	BD Postos	

Tabela 7.1: Dados para as análises A1 e A2.

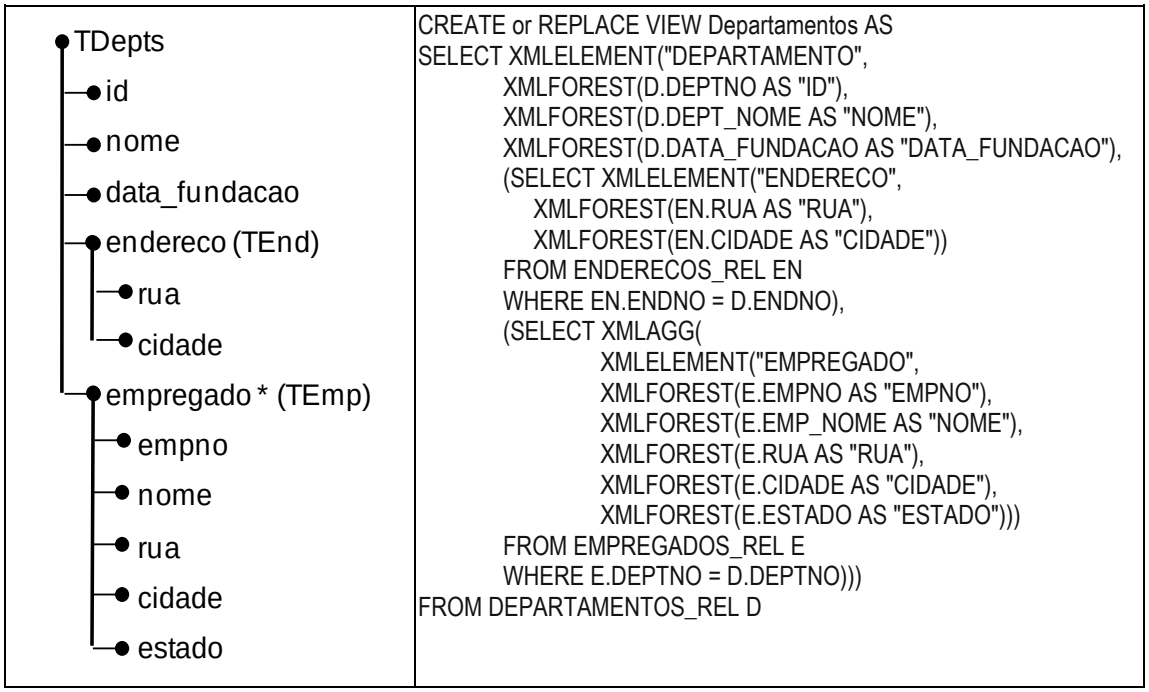
A seguir, a subseção 7.4.1 apresenta uma avaliação comparativa do processamento de consultas XQuery do ***RelP*** e do SGBD Oracle 10g, e a subseção 7.4.2 apresenta uma avaliação comparativa do processamento de consultas WFS do ***RelP***, do Deegree WFS v. 2.2.1 e do GeoServer v. 1.7.4.

7.4.1 Avaliação de consultas XQuery

O objetivo desta análise consiste em avaliar a eficiência da tradução de consultas XQuery em consultas SQL/XML no *framework* ***RelP*** e no SGBD Oracle 10g.

Para avaliarmos a tradução de consultas XQuery no Oracle 10g, criamos a visão XML Departamentos a partir do esquema do banco de dados Empresa, apresentado na Figura 7.1. A Figura 7.26(a) mostra o esquema da visão Departamentos, cujo elemento primário é Depts do tipo TDepts. A visão Departamentos foi criada no Oracle tal como apresentada na Figura 7.26(b). No Oracle, as consultas XQuery sobre a visão Departamentos foram criadas utilizando a função XMLQuery [25].

Para avaliarmos a tradução de consultas XQuery no ***RelP***, criamos a visão XML Departamentos, apresentada na Figura 7.26(a), conforme os passos descritos no Capítulo 4. As consultas XQuery foram construídas através do componente *XQuery Browser*, baseando-se nos passos apresentados no Capítulo 5.



(a) (b)

Figura 7.26: (a) Esquema da visão XML Departamentos. (b) Consulta SQL/XML para criar a visão XML Departamentos.

Para a análise, foram definidas 20 consultas XQuery com base na estrutura das consultas encontradas em diversas aplicações XML, como mostra o Anexo B. Essas consultas possuem diferentes níveis de complexidade: apresentam expressões de caminhos de tamanho variado, com uma ou mais cláusulas de filtro e com ou sem reconstrução de elementos.

A Figura 7.27 e a tabela 7.2 mostram o gráfico e os dados, respectivamente, dos 20 tempos de resposta das consultas XQuery processadas no Oracle 10g e no ***RelP***. Os experimentos mostraram que as consultas XQuery que apresentam mais de duas expressões de caminho e, cada caminho sendo maior ou igual a três (consultas de 5 a 12 e consulta 14) foram executadas em menor tempo no ***RelP*** do que no Oracle 10g. Além disso, as consultas que apresentam reconstrução de elementos (consulta 13 e consulta de 15 a 20), os tempos de resposta no ***RelP*** e no Oracle 10g são semelhantes.

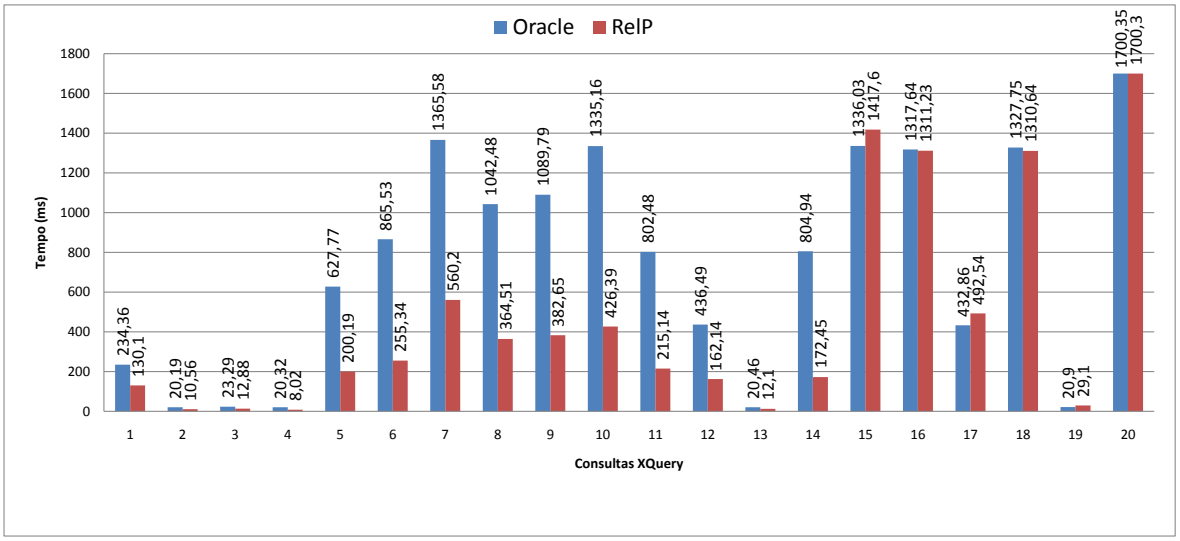


Figura 7.27: Gráfico dos tempos de resposta por consulta XQuery no Oracle 10g e ***RelP***.

Consultas XQuery	Oracle 10g (ms)	RelP (ms)
1	234,36	130,1
2	20,19	10,56
3	23,29	12,88
4	20,32	8,02
5	627,77	200,19
6	865,53	255,34
7	1365,58	560,2
8	1042,48	364,51
9	1089,79	382,65
10	1335,16	426,39
11	802,48	215,14
12	436,49	162,14
13	20,46	12,1
14	804,94	172,45
15	1336,03	1417,6
16	1317,64	1311,23
17	432,86	492,54
18	1327,75	1310,64
19	20,9	29,1
20	1700,35	1700,3

Tabela 7.2: Tempos de resposta por consulta XQuery no Oracle 10g e ***RelP***.

No Oracle, toda consulta XQuery a ser processada no banco de dados é traduzida primeiramente em uma consulta SQL/XML equivalente. Em um dos passos deste processo, a consulta XQuery é traduzida em uma árvore de consulta, chamada XQuery Expression Tree. Em seguida, o compilador percorre essa árvore gerando a

consulta SQL/XML. No ***RelP***, por sua vez, o processo de tradução de consultas é feito com o auxílio dos Templates de Consulta (TCs) da visão, os quais são fragmentos de consultas SQL/XML gerados a partir das Assertivas de Correspondência da visão. O uso dos TCs simplifica a tradução de consulta, uma vez que todo o processamento dos mapeamentos complexos da visão é realizado em tempo de projeto. Deste modo, a reescrita da consulta se resume a uma composição dos fragmentos relevantes para construir a consulta SQL/XML correta.

Portanto, os resultados apresentados mostram que o ***RelP*** apresenta alta performance de processamento de consultas XQuery, se mostrando como uma excelente alternativa para solução em SGBDs que não possuem mecanismo de visão XML. Entretanto, o Oracle 10g suporta uma maior diversidade de consultas XQuery e, conseqüentemente, permite traduções de consultas XQuery mais complexas.

7.4.2 Avaliação de consultas WFS

O objetivo desta análise consiste em avaliar a eficiência de tradução de requisições WFS de consulta no ***RelP***, Deegree WFS e GeoServer.

No GeoServer, as visões GML são simples, pois não permitem navegar através de chaves estrangeiras para obter dados em outras tabelas. Dessa forma, realizamos dois testes distintos. Primeiro apresentamos uma análise comparativa do ***RelP*** e do Deegree WFS, e depois mostramos uma análise comparativa do ***RelP*** e do GeoServer.

Análise com Deegree WFS e *RelP*

Para avaliarmos a tradução de consultas WFS no Deegree WFS, publicamos a visão GML F_Posto sobre o banco Postos, cujo esquema é descrito na Figura 6.2. A visão F_Posto foi criada no Deegree WFS baseando-se nos passos apresentados na Seção 7.3. As consultas WFS sobre a visão GML foram executadas através da interface gráfica de consulta disponibilizada no Deegree WFS.

Para avaliarmos a tradução de consultas WFS no ***RelP***, criamos a visão GML F_Posto, apresentada na Figura 6.2, conforme os passos descritos no Capítulo 6.

As consultas WFS, no ***RelP***, foram construídas através do componente *XQuery Browser*, baseando-se nos passos apresentados no Capítulo 5.

No Deegree WFS e no ***RelP*** foram executadas 20 consultas WFS sobre as visões F_Posto. As consultas WFS, descritas no Anexo C, apresentam expressões caminhos e aninhamentos com ou sem elementos de construção e/ou filtros de comparação, lógicos e/ou espaciais.

A Figura 7.28 e a tabela 7.3 mostram o gráfico e os dados, respectivamente, dos 20 tempos de resposta das traduções de consultas processadas no Deegree WFS e no ***RelP***.

O experimentos mostram que todas as consultas WFS são executadas em menos tempo no ***RelP***, principalmente nas consultas que apresentam expressões de caminho e onde existe uma quantidade maior de dados a serem retornados. No Deegree WFS, a eficiência de tradução de consulta é menor, pois as expressões de caminho podem gerar mais de uma consulta SQL, e, conseqüentemente, mais resultados SQL são gerados para serem compilados em um único documento GML. No ***RelP***, uma única consulta SQL/XML é sempre gerada e o resultado XML é compilado pelo próprio SGBD.

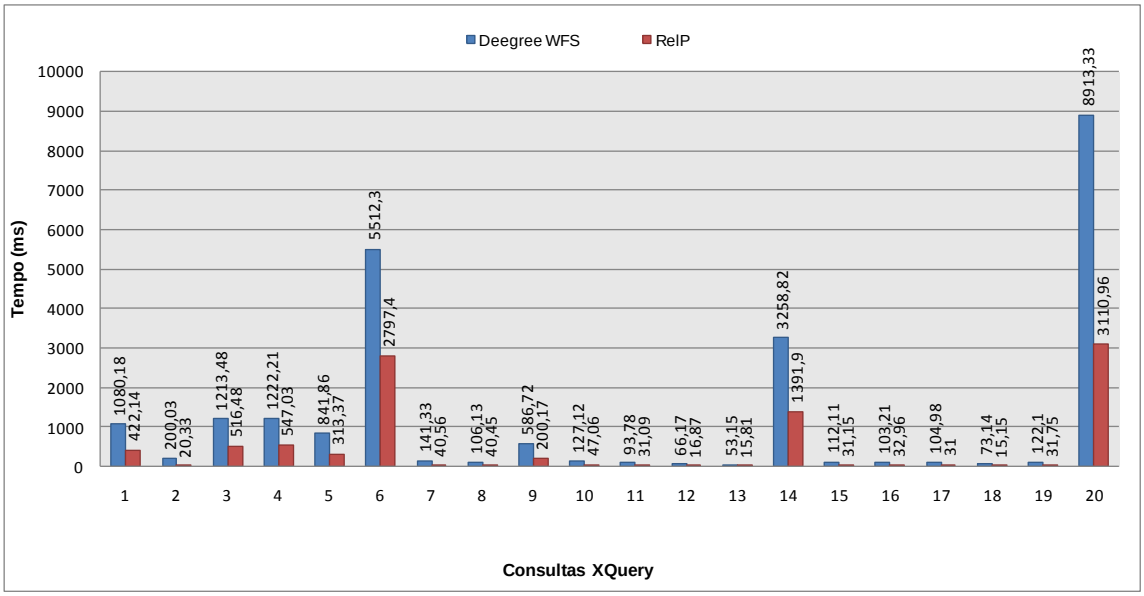


Figura 7.28: Gráfico dos tempos de resposta por consulta WFS no Deegree WFS e ***RelP***.

Consultas WFS	Deegree WFS (ms)	RelP (ms)
1	1080,18	422,14
2	200,03	20,33
3	1213,48	516,48
4	1222,21	547,03
5	841,86	313,37
6	5512,3	2797,4
7	141,33	40,56
8	106,13	40,45
9	586,72	200,17
10	127,12	47,06
11	93,78	31,09
12	66,17	16,87
13	53,15	15,81
14	3258,82	1391,9
15	112,11	31,15
16	103,21	32,96
17	104,98	31
18	73,14	15,15
19	122,1	31,75
20	8913,33	3110,96

Tabela 7.3: Tempos de resposta por consulta WFS no Deegree WFS e ***RelP***.

Análise com GeoServer e *RelP*

Para avaliarmos a tradução de consultas WFS no GeoServer, publicamos a visão GML F_Posto' sobre o banco Postos (Figura 6.1). A Figura 7.29 mostra o esquema da visão F_Posto', cujo elemento primário é F_Posto do tipo TF_Posto. A visão F_Posto' foi criada no GeoServer conforme os passos apresentados na Seção 7.3. As consultas WFS, sobre a visão F_Posto', foram executadas através da interface de consulta disponibilizada no GeoServer.

Para avaliarmos a tradução de consultas WFS no ***RelP***, criamos a visão GML F_Posto', apresentada na Figura 7.29, conforme os passos descritos no Capítulo 6. As consultas WFS, no ***RelP***, foram construídas através do componente *XQuery Browser*, baseando-se nos passos apresentados no Capítulo 5.

No GeoServer e no ***RelP*** foram executadas 20 consultas WFS sobre suas visões F_Posto'. As consultas WFS, que estão no Anexo D deste trabalho, apresentam expressões sem caminho e/ou filtros comparativos, lógicos e/ou espaciais.

A Figura 7.30 e tabela 7.4 mostram o gráfico e os dados, respectivamente, dos 20 tempos de resposta das traduções de consultas processadas no GeoServer e no ***RelP***. Todas as consultas WFS avaliadas são executadas em menos tempo no ***RelP***. Em

resumo, no GeoServer, o tempo de tradução de uma consulta é maior do que no ***RelP*** devido ao tempo gasto para gerar o documento GML a partir do resultado SQL (*tagging*). No ***RelP***, esse documento é gerado pelo próprio SGBD.

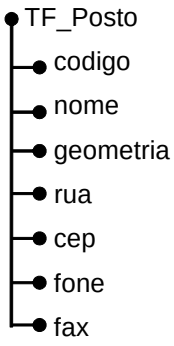


Figura 7.29: Esquema da visão GML F_Posto'.

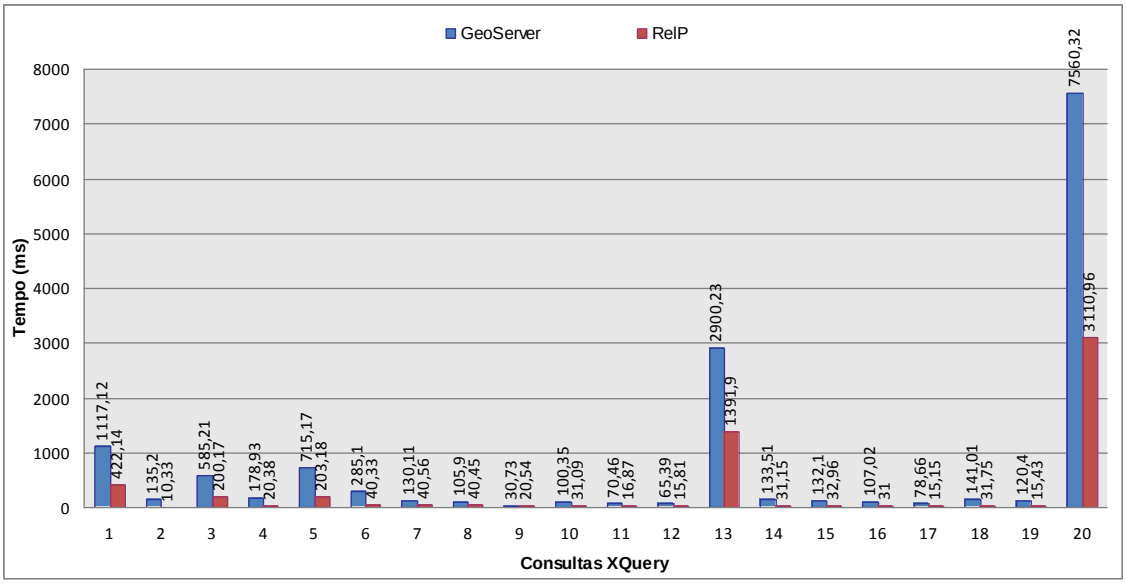


Figura 7.30: Gráfico dos tempos de resposta por consulta WFS no GeoServer e ***RelP***.

Consultas WFS	GeoServer (ms)	RelP (ms)
1	1117,12	422,14
2	135,2	10,33
3	585,21	200,17
4	178,93	20,38
5	715,17	203,18
6	285,1	40,33
7	130,11	40,56
8	105,9	40,45
9	30,73	20,54
10	100,35	31,09

11	70,46	16,87
12	65,39	15,81
13	2900,23	1391,9
14	133,51	31,15
15	132,1	32,96
16	107,02	31
17	78,66	15,15
18	141,01	31,75
19	120,4	15,43
20	7560,32	3110,96

Tabela 7.4: Tempos de resposta por consulta WFS no GeoServer e ***RelP***.

7.5 Conclusão

Neste capítulo, apresentamos as principais propostas de publicação e consulta de dados relacionais através de visões XML e GML, e discutimos a análise da eficiência do processamento de consultas do ***RelP*** comparando-o com Oracle 10g, Deegree WFS e GeoServer.

Em síntese, as propostas: XTables, SilkRoute e Deegree WFS; não garantem um processamento eficiente de consulta pois mais de uma consulta SQL pode ser gerada. No Deegree WFS, a publicação de visões GML é complexa e exige conhecimentos de um especialista em XML e nas especificações do OGC. No GeoServer, uma visão GML só pode extrair dados de uma única tabela. No *Framework de Três Fases* do Clio, não é possível realizar consultas sobre os esquemas. Finalmente, dentre os SGBDs, *Oracle* e o DB2 são os únicos SGBDs que suportam a criação e consulta de visões XML utilizando o padrão SQL/XML e a linguagem XQuery.

Na avaliação de desempenho, foi verificado que o ***RelP*** apresenta um mecanismo de tradução de consultas bastante eficiente. Na análise ***A1***, os resultados apresentados mostram que o ***RelP*** apresenta eficiência semelhante ao Oracle, se mostrando como uma excelente alternativa para solução em SGBDs que não possuem mecanismo de visão XML. Na análise ***A2***, todas as consultas WFS avaliadas são executadas em menos tempo no ***RelP*** do que no Deegree WFS e no GeoServer. Neste caso, a utilização da linguagem SQL/XML torna o processo de tradução mais rápido no ***RelP***, pois o resultado XML é processado pelo próprio SGBD, e não pelo sistema, como ocorre com no Deegree WFS e o GeoServer.

Dessa forma, a eficiência de tradução de consultas no ***RelP*** se deve principalmente a utilização dos Templates de Consulta e a utilização da linguagem SQL/XML. O uso dos TCs simplifica a tradução de consulta, que se resume uma composição dos fragmentos relevantes para construir a consulta SQL/XML correta.

Capítulo 8
Conclusão

Neste trabalho, apresentamos um *middleware*, chamado ***RelP***, para publicação e consulta de dados relacionais através de visões SQL/XML. Como visto, o ***RelP*** atende os três requisitos de um *framework* de publicação de dados relacionais no formato XML: (1) *Geral*, pois o formalismo das Assertivas de Correspondência permite definir diversos tipos de mapeamentos, inclusive para esquemas com estruturas diferentes; (2) *Seletivo*, pois permite a definição de consultas XQuery sobre uma visão XML virtual e garante que somente os dados relevantes às consultas serão obtidos da fonte de dados; e (3) *Eficiente*, pois, como apresentado no Capítulo 7, o ***RelP*** apresenta um mecanismo de tradução de consultas bastante eficiente.

Em nossa abordagem, utilizamos *serviços Web de acesso a dados*, pois oferecem uma forma transparente e flexível para a publicação de dados, como também encapsulam detalhes de acesso e recuperação entre aplicações e dados. Dessa forma, propomos a especificação *Web XML Service (WXS)* (vide Capítulo 3), baseada na especificação *Web Feature Service (WFS)*, a qual oferece uma interface padrão para consulta de dados XML na Web. Implementamos o serviço *WXService*, o qual fornece os seguintes métodos da especificação WXS: *getCapabilities*, *getViewType* e *query*.

No Capítulo 4, apresentamos a ferramenta *XML View Publisher* (XVP) para publicação de visões XML no ***RelP***. A ferramenta gera automaticamente os Templates de Consulta a partir das ACs que são definidas através da interface gráfica. Os TCs são automaticamente gerados através do algoritmo **GeraTC**.

No ***RelP***, consultar uma visão XML consiste em enviar uma consulta XQuery sobre o esquema da visão através do método query do serviço Web *WXService*. O usuário que não domina a linguagem XQuery pode utilizar a ferramenta *XQuery Browser*, apresentada no Capítulo 5, para construir intuitivamente consultas XQuery. No Capítulo 5, apresentamos também os algoritmos **GeraTPQX** e **GeraSQL/XML** responsáveis pela tradução de consultas XQuery em consultas SQL/XML no ***RelP***.

No Capítulo 6, apresentamos uma adaptação do nosso *middleware* à especificação WFS, de modo que é possível publicar e consultar visões GML sobre uma base de dados relacional. Para tanto, implementamos o serviço *WFService*, o qual é uma adaptação do serviço *WXService* com base na especificação WFS. O algoritmo de

tradução de consultas **GeraSQL/XML** foi modificado para traduzir consultas WFS sobre visões GML em consultas SQL/XML sobre o banco de dados. Foi adaptado também a ferramenta *XML View Publisher*, para tratar visões GML, e a ferramenta *XQuery Browser*, para criar consultas WFS de forma gráfica.

No Capítulo 7, foi feito um estudo sobre a eficiência do processamento de consultas do ***RelP***. Mostramos que o ***RelP*** traduz eficientemente consultas XQuery e consultas WFS, e se mostra como uma excelente alternativa para solução em SGBDs que não possuem mecanismo de visão XML. Esta eficiência é devida, principalmente, ao uso dos Templates de Consulta. Os TCs simplificam o processo de tradução de consulta, uma vez que todo o processamento dos mapeamentos complexos da visão é realizado em tempo de projeto. Deste modo, a reescrita da consulta se resume a uma composição dos fragmentos relevantes para construir a consulta SQL/XML correta.

Além disso, a estratégia de tradução de consultas para SQL/XML se mostra vantajosa, pois todo o processamento de transformação de dados relacionais para o formato XML é feito diretamente pelo SGBD ao mesmo tempo em que a consulta é executada. Outra vantagem do nosso enfoque é que oferecemos um ambiente unificado para a utilização de *serviços Web de Acesso a Dados* (publicação de visões XML e GML), até então, não oferecido por sistemas relacionados.

8.1 Perspectivas Futuras

A seguir, descrevemos alguns aspectos que não foram contemplados por este trabalho, mas podem ser explorados em trabalhos futuros:

1. Aumento da expressividade de consultas XQuery aceitas pelo ***RelP***, possibilitando a tradução de consultas que não podem ser descritas usando a *Tree Pattern Query*.
2. Atualização dos algoritmos de tradução de consultas do ***RelP*** para permitir traduções de consultas para a linguagem SQL/XML 2006 [50]. A SQL/XML:2006 adiciona novas funções de publicação/expressões, permitindo que os usuários definam consultas mais complexas. Maiores detalhes sobre as novas funções de publicação/expressões podem ser vistos em [50];
3. Extensão dos componentes e interfaces do ***RelP*** para permitir o processo de atualização de dados relacionais a partir de visões XML e GML usando serviços Web.

Referências Bibliográficas

[1] Shanmugasundaram, J., Shekita, E., Barr, R., Carey, M., Lindsay, B., Pirahesh, H., and Reinwald, B. (2001) "Efficiently publishing relational data as XML documents", In *Proceeding of 27th Very Large Database Conference*, Rome, Italy, p. 133-154.

[2] Shanmugasundaram, J. et al. (2001) "Querying XML Views of Relational Data", In *Proceeding of 27th Very Large Database Conference*, Rome, Italy, p. 261–270.

[3] Adams, D. (2005) "Oracle® XML DB 10g Release 2 Developer's Guide", http://www.oracle.com/technology/documentation/database10gR2.html.

[4] Oracle Corporation – Oracle XML Developer's Kit (Acesso 2008) http://www.oracle.com/technology/tech/xml/xdkhome.html.

[5] IBM Corporation DB2. (2009) "pureXML Guide", ftp://ftp.software.ibm.com/ps/products/db2/info/vr97/pdf/en_US/DB2pureXML-db2xge970.pdf, Outubro.

[6] PostgreSQL Documentation (2009), http://www.postgresql.org/docs/8.4/static/functions-xml.html, Setembro.

[7] Jiang, H., Ho, H., Popa, L., and Han, W. (2007) "Mapping-driven XML transformation", In *Proceedings of the 16th international Conference on World Wide Web*, Canada.

[8] Miller, R. J. (2007) "Retrospective on Clio: Schema Mapping and Data Exchange in Practice", In *Proceedings of the 2007 International Workshop on Description Logics*, Vol. 250, Italy.

[9] Haas, L. M., Fernandez, M., Ho, H., Popa L., and Roth, M. (2005) "Clio grows up: From research prototype to industrial tool", In *ACM SIGMOD*, p. 805–810.

[10] Hernandez, M. A., Miller, R. J., Haas, L., Yan, L., Ho, C. T. H., and Tian, X. (2001) "Clio: A semi-automatic tool for schema mapping", In *International Conference on Management of Data*, *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, p. 607–612.

[11] Ré, C., Brinkley, J., and Suciu, D. (2006) "A Performant XQuery to SQL Translator", Technical Report.

[12] Fernandes, M., and Simeon, J. (Acesso 2009) "Galax: the XQuery implementation for discriminating hackers", https://launchpad.net/ubuntu/+source/galax/1.1-1.

[13] Fernández, M., Kadiyska, Y., Suciu, D., Morishima, A., and Tan, W. (2002) "SilkRoute: A framework for publishing relational data in XML", In *ACM Transactions on Data-base Systems (TODS)*, Vol. 27, p. 438–493.

[14] Fernández, M., Morishima, A., and Suciu, D. (2001) "Publishing Relational Data in XML: the SilkRoute Approach*"*, In *Institute of Electrical and Electronics Engineers, Inc*.

[15] Fernández, M., Tan, W., and Suciu, D. (2000) "Silkroute: Trading between relations and XML", In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, Netherlands.

[16] Melnik, S., Bernstein, P. A., Halevy, A. Y., and Rahm, E. (2005) "Supporting Executable Mappings in Model Management", In *SIGMOD Conference*, p. 167–178.

[17] Santos, L. A. de L. (2004) "XML Publisher: Um Framework para Publicar Dados Objeto Relacionais em XML" [Dissertação], Departamento de Computação, Universidade Federal do Ceará, Fortaleza-CE.

[18] Funderburk, J. E. et al. (2002) "XTABLES: Bridging relational technology and XML", *IBM Systems Journal*, Vol. 41, Nº 4.

[19] Shanmugasundaram, J. et al. (2001) "XPERANTO: Bridging Relational Technology and XML", *IBM Research Report*.

[20] Carey, M. J., Kiernan, J., Shanmugasundaram, J., Shekita, E. J., and Subramanian, S. N. (2000) "XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents", In *Proceedings of the 26th International Conference on Very Large Data Bases*, p. 646–648.

[21] Carey, M. et al. (2000) "XPERANTO: Publishing object-relational data as XML", In *Proceedings Third International Workshop on the Web and Databases*, Dallas, Texas, p. 105-110.

[22] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F. (2004) "SQL:2003 has been published", In *ACM SIGMOD*, Vol. 33, p. 119–126.

[23] Eisenberg, A. and Melton, J. (2001) "SQL/XML and the SQLX Informal Group of Companies", In *ACM SIGMOD*, Vol. 30.

[24] Krishnaprasad, M., Liu, Z. H., Manikutty, A., Warner, J. W., Arora, V., and Kotsolo-vos, S. (2004) "Query Rewrite for XML in Oracle XML DB", In *Proceedings of the 30th International Conference on Very Large Data Bases*, p. 1122–1133.

[25] Liu, Z. H., Krishnaprasad, M., and Arora, V. (2005) "Native XQuery Processing in Oracle XMLDB", In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p. 828–833.

[26] Muench, S. (2000) "Building Oracle XML Applications", O'Reilly.

[27] Murthy, R., and Banerjee, S. (2003) "XML Schemas in Oracle XML DB", In *Proceedings of the 29th International Conference on Very Large Data Bases*, p. 1009–1018.

[28] Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., and Fagin, R. (2002) "Translating Web Data", In *Proceedings of the International Conference on Very Large Data Bases*, p. 598–609.

[29] Araujo, M. A., (2005) "Web Services na Informação Geográfica" [Dissertação], Universidade do Minho, Braga, Portugal.

[30] World Wide Web Consortium (Acesso 2009), Web Services, http://www.w3.org/2002/ws/

[31] World Wide Web Consortium (Acesso 2009), Simple Object Access Protocol (SOAP) 1.2, http://www.w3.org/TR/soap/

[32] World Wide Web Consortium (Acesso 2009), Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

[33] Bryan, D., Draluk, V., Ehnebuske, D., Glover, T., Hately A., and Husband, Y. L. (2002) "UDDI version 2.04 API specification", http: //uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.

[34] Advancing Open Standards for the Information Society – OASIS (Acesso 2009), http://www.oasis-open.org/home/index.php.

[35] OpenGIS Consortium (Acesso 2008), OpenGIS Web Services Architecture, http://www.opengis.org.

[36] OpenGIS Consortium (Acesso 2008), Schema for Geography Markup Language (GML). http://www.opengeospatial.org/standards/gml.

[37] OpenGIS Consortium (Acesso 2008), OGC Reference Model, http://www.opengeospatial.org/standards/orm,

[38] OpenGIS Consortium (Acesso 2009), Web Feature Service Implementation Specification (WFS), http://www.opengeospatial.org/standards/wfs.

[39] OpenGIS Consortium (Acesso 2009), Filter Encoding Implementation Specification, http://www.opengeospatial.org/standards/filter.

[40] World Wide Web Consortium (Acesso 2008), XQuery 1.0: An XML Query Language. W3C Recommendation, http://www.w3.org/TR/xquery/.

[41] World Wide Web Consortium (Acesso 2008), XML Path Language (XPath) Version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath.

[42] World Wide Web Consortium (Acesso 2008), XSL Transformations (XSLT) Version 1.0, W3C Recommendation, http://www.w3.org/TR/xslt.

[43] Benham, S.E. (2003) "XML-Enabled Data Management Product Architecture and Technology", In *XML Data Management, Native XML and XML-Enable Database Systems*, Chaudhri, A.B., Rashid, A., and Zicari, R. (eds.), Addison Wesley.

[44] Deegree (Acesso 2009), http://www.deegree.org/.

[45] Deegree WFS – Versão 2.2.1 (Acesso 2009), http://deegree.sourceforge.net.

[46] DeegreeTools (Acesso 2009), https://wiki.deegree.org/deegreeWiki/deegreeTools.

[47] GeoServer (Acesso 2009), http://geoserver.org/.

[48] Rys, M. (2003) "XML Support in Microsoft SQL Server 2000", In *XML Data Management, Native XML and XML-Enable Database Systems*, Chaudhri, A.B., Rashid, A., and Zicari, R. (eds.), Addison Wesley.

[49] MySQL Documentation (2009), http://dev.mysql.com/doc/refman/5.1/en/xml-functions.html, Setembro.

[50] PostgreSQL - XML Support (2009), http://developer.postgresql.org/index.php/XML_Support#SQL.2FXML_2006.

[51] Fagin, R., Kolaitis, P. G., Miller, R. J. and Popa, L. (2003) "Data Exchange: Semantics and Query Answering", In *Lecture Notes in Computer Science*, Vol. 2572, *Proceedings of the 9th International Conference on Database Theory*, p. 207-224.

[52] Funderburk, J. (2003) "XML for Tables", IBM Research and IBM Academic initiative, http://www.alphaworks.ibm.com/tech/xtable.

[53] Costa, C. M., Vidal, V. M. P., Lemos, F. C. (2009) "WXP: Um Framework para geração de serviços Web de acesso a dados relacionais e geográficos", In *InfoBrasil*.

[54] Vidal, V.M.P., and Lemos, F.C. (2006) "Automatic Generation of SQL/XML Views", In *Proceedings of 21st Brazilian Symposium on Databases*, p. 221–235.

[55] Vidal, V. M. P., Araujo, V. S., and Casanova, M. A. (2005) "Towards Automatic Generation of Rules for the Incremental Maintenance of XML Views over Relational Data", In *Web Information Systems Engineering*, p. 189–202.

[56] Vidal, V.M.P., Casanova, M.A., and Lemos, F.C. (2005) "Automatic Generation of XML Views of Relational Data", Technical Report, http://arida.mdcc.ufc.br/, Universidade Federal do Ceará.

[57] Vidal, V.M.P., Santos, L.A.L., e Lemos, F.C. (2005) "Geração Automática de Visões de Objeto de Dados Relacionais", In *Proceedings of 20th Brazilian Symposium on Databases*, p. 160–174.

[58] Vidal, V. M. P., Teixeira, G. M. L., and Feitosa, F. B. (2004) "Towards Automatic Feature Type Publication", In *VI Brazilian Symposium on GeoInformatics*.

[59] Vidal, V. M. P., Santos, L. A. L., Oliveira, W. G. C., Araújo, V. S., Lemos, F. C., and Lima, D. R. C., (2004) "XML Publisher: Um Framework para Publicação de Dados Armazenados em Banco de Dados Relacional ou Objeto Relacional como XML", In *1ª Seção de Demos, Simpósio Brasileiro de Banco de Dados*, p. 07–12.

[60] Yu, C., and Popa, L., (2004) “Constraint-Based XML Query Rewriting For Data Integration”, In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of data*, SESSION: Research sessions: Data Integration, p. 371–382.

[61] Bernstein, P. A., and Melnik, S. (2007) “Model management 2.0: manipulating richer mappings”, In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 1–12.

[62] Kozlova I., Ritter N., and Reimer O. (2006) “Towards Integrated Query Processing for Object-Relational and XML Data Sources”, In *Proceedings of the 10th International Database Engineering & Application Symposium*, *IEEE Computer Society*, p. 295–300.

[63] Lima, P. (2002) “Intercâmbio de Dados Espaciais: Modelos, Formatos e Conversores” [Dissertação], Instituto Nacional de Pesquisas Espaciais - INPE.

[64] Paiva, V. M. (2007) “Banco de Dados Geográficos: Estudo de Caso da Aplicação das Extensões” [Monografia], Universidade Federal de Juiz de Fora, Juiz de Fora, MG.

[65] Silva, R. A. B. (2004) “Interoperabilidade na Representação de Dados Geográficos: GEOBR e GML 3.0 no Contexto da Realidade dos Dados Geográficos no Brasil” [Dissertação], Instituto Nacional de Pesquisas Espaciais - INPE.

[66] Papakonstantinou, Y. and Vassalos, V. (1999) “Query rewriting for semistructured data”, In *Proceedings of the 1999 ACM SIGMOD international Conference on Management of Data*, p. 455–466, Philadelphia, Pennsylvania, USA.

[67] Krishnamurthy, R., Kaushik, R., and Naughton, J. (2003) “XML-to-SQL Query Translation Literature: The State of the Art and Open Problems”, In *Proceedings of the 1st International XML Database Symposium*, p. 1–18, Berlin, Germany.

[68] DeHaan, D., Toman, D., Consens, M. P., and Özsu, M. T. (2003) “A comprehensive XQuery to SQL translation using dynamic interval encoding”, In

Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data, p. 623–634, San Diego, California.

[69] Grust, T., Sakr, S., and Teubner, J. (2004) "XQuery on SQL hosts", In *Proceedings of the Thirtieth international Conference on Very Large Data Bases*, Vol. 30, p. 252-263, Toronto, Canada.

[70] Krishnamurthy, R., (2004) "XML-to-SQL Query Translation" [Tese], University of Wisconsin, Madison, USA.

[71] Chen, Y. (2004) "XQuery Query Processing in Relational Systems" [Tese], University of Waterloo, Ontario, Canada.

[72] World Wide Web Consortium (Acesso 2009), Extensible Markup Language (XML), W3C Recommendation, http://www.w3.org/XML.

[73] World Wide Web Consortium (Acesso 2009), XML Schema, W3C Recommendation, http://www.w3.org/XML/Schema.

[74] Stylus Studio XML Enterprise Suite (Acesso 2009), http://www.stylusstudio.com.

[75] Lee, G. (2007) "Oracle® XML DB 11g Technical Overview", http://www.oracle.com/technology/tech/xml/xmldb/Current/xmldb_11g_twp.pdf

[76] FROZZA, A. A., MELLO, R. dos S. (2006) "Determinando equivalências semânticas entre componentes de Esquemas GML", In *Escola Regional de Banco de Dados - ERBD*, p. 63 – 68.

Anexo A

Geração dos arquivos: AC.xml e TC.xml, no *RelP*

O esquema do arquivo das Assertivas de Correspondência (ACs), AC.xml, é apresentado na Figura A.1. O arquivo armazena para cada visão XML ou GML as seguintes informações:

i. O tipo da assertiva, nome e tipo da visão, nome e tipo da tabela pivô;
ii. Um conjunto de Assertivas de Correspondência. Todo elemento, atributo ou tipo da visão tem uma Assertiva de Correspondência e cada AC possui informações da propriedade da visão e informações do esquema do banco de dados:
    a. As informações da propriedade da visão são: o tipo do elemento pai da propriedade, nome da propriedade e seu tipo, se a propriedade é do tipo complexo, se a propriedade tem varias ocorrências e se a propriedade é chave;
    b. As informações do esquema do banco de dados são: nome do atributo ou da chave estrangeira, tipo, se é complexo, se tem varias ocorrências, se é chave primária e se é chave estrangeira. As informações de uma chave estrangeira são: nome das duas tabelas envolvidas e seus respectivos atributos.

O ***RelP*** também gera o arquivo dos Templates de Consulta (TCs), TC.xml. Este é utilizado para traduzir uma consulta XQuery ou uma requisição WFS de consulta em uma consulta SQL/XML. O esquema do documento TC.xml é apresentado na Figura A.2 e para cada visão XML ou GML, ele armazena as seguintes informações:

iii. Nome da visão publicada, nome da tabela pivô do esquema do banco de dados, tipo de tradução;
iv. Um conjunto de expressões de caminho da visão. Uma expressão de caminho corresponde a um elemento ou tipo na visão e é composto de:
    a. Conteúdo da expressão de caminho;

b. Nome da tabela e nome do atributo no esquema do banco de dados correspondente a expressão de caminho, caso exista;

c. Template SQL/XML para a expressão de caminho, que é composto de:

   i. Cláusula select;

   ii. Um conjunto de elementos caso a expressão de caminho seja do tipo complexo;

   iii. Clausula from, se necessário, que contém um conjunto de tabelas relacionadas com tabela pivô no esquema do banco de dados;

   iv. Clausula where, se necessário, que contém um conjunto de expressão condicional.

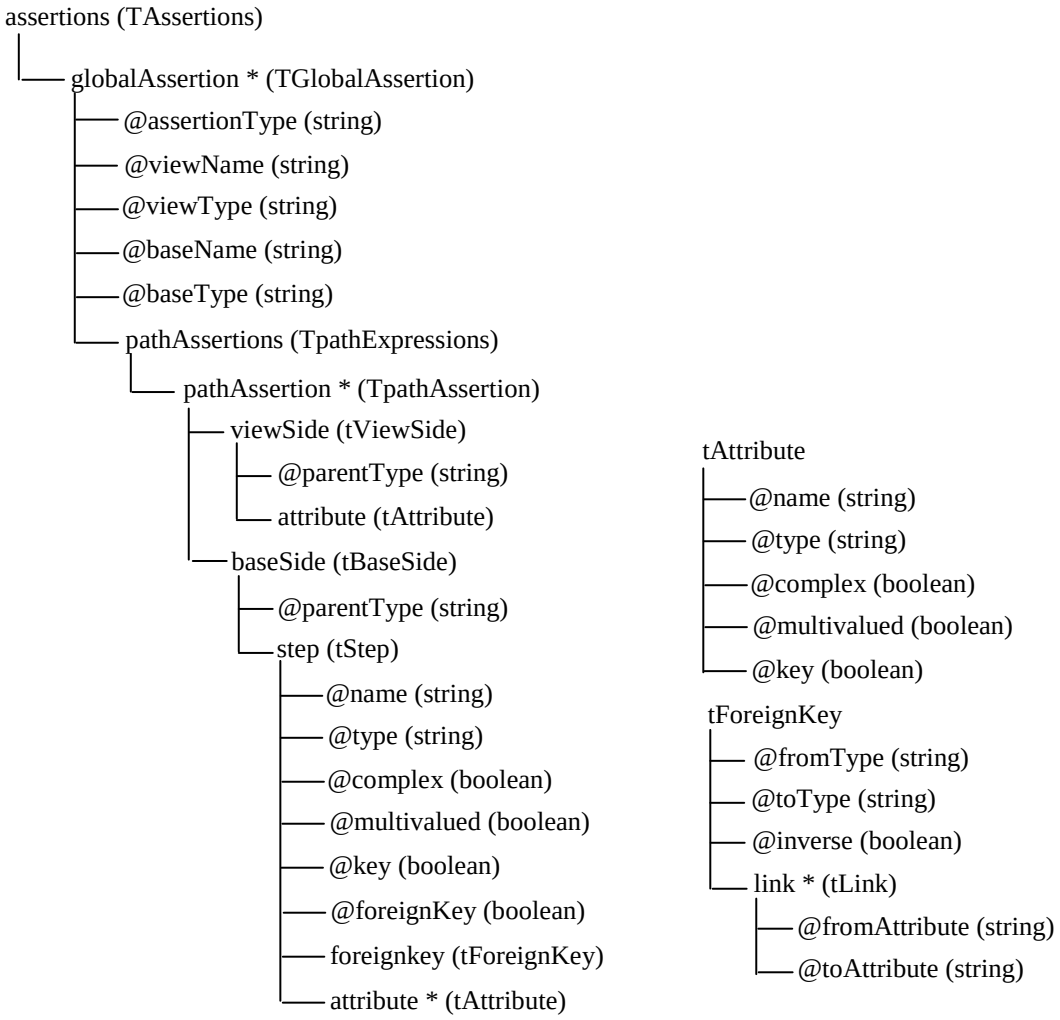


Figura A.1: Esquema do arquivo das Assertivas de Correspondência, AC.xml.

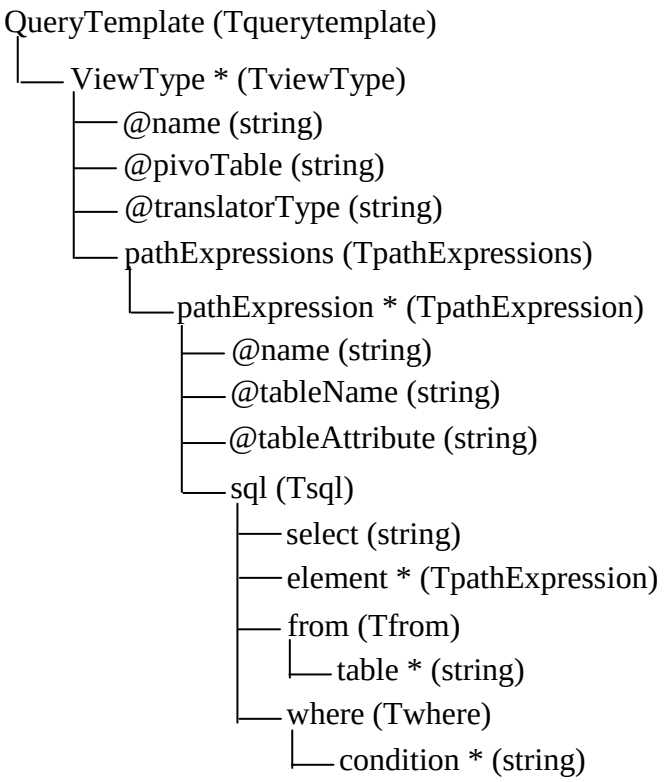


Figura A.2: Esquema do arquivo dos Templates de Consulta, TC.xml.

Anexo B

Consultas XQuery executadas no *RelP* e no Oracle 10g

1) Retorna todos os departamentos juntamente com todos os seus empregados.

```
for $d in /Departamentos
return <Departamentos>{
                $d/Departamento
        }</Departamentos>
```

2) Retorna o código e a data de fundação dos departamentos que tem o empregado de código igual a 250.

```
for $d in /Departamentos
where $d/Departamento/Empregado/empno = 250
return <Departamentos>{
                $d/Departamento/id,
                $d/Departamento/data_fundacao
      }</Departamentos>
```

3) Retorna o código, o nome e a data de fundação do departamento fundado em 10/05/2009.

```
for $d in /Departamentos
where $d/Departamento/data_fundacao = xs:date("10/05/2009")
return <Departamentos>{
                $d/Departamento/id,
                  $d/Departamento/nome,
                  $d/Departamento/data_fundacao
      }</Departamentos>
```

4) Retorna o código e os empregados de todos os departamentos.

```
for $d in /Departamentos
return <Departamentos>{
                $d/Departamento/id,
                 $d/Departamento/Empregado
        }</Departamentos>
```

5) Retorna o código do departamento juntamente com código e nome de todos os empregados.

```
for $d in /Departamentos
return <Departamentos>{
                $d/Departamento/id,
                  <Empregados>{
                          $d/Departamento/Empregado/empno,
                          $d/Departamento/Empregado/nome
                  }</Empregados>
        }</Departamentos>
```

6) Retorna o código do departamento seguido do código e nome do empregado cujo nome é “João Fernando”.

```
for $d in /Departamentos
where $d/Departamento/Empregado/nome = "Joao Fernando"
return <Departamentos>{
                    $d/Departamento/id,
                      <Empregados>{
                                $d/Departamento/Empregado/empno,
                                $d/Departamento/Empregado/nome
                      }</Empregados>
          }</Departamentos>
```

7) Retorna o código dos departamentos juntamente com o nome e a rua do empregado com código igual a 1313 e nome igual a “Pedro Davi”.

```
for $d in /Departamentos
where $d/Departamento/id = 1313 and $d/Departamento/Empregado/nome = "Pedro Davi"
return <Departamentos>{
                    $d/Departamento/id,
                      <Empregados>{
                                $d/Departamento/Empregado/nome,
                                $d/Departamento/Empregado/rua
                      }</Empregados>
          }</Departamentos>
```

8) Retorna o código dos departamentos juntamente com nome, rua e cidade de seus empregados.

```
for $d in /Departamentos
return <Departamentos>{
                    $d/Departamento/id,
                      $d/Departamento/Empregado/nome,
                      $d/Departamento/Empregado/rua,
                      $d//Empregados/Empregado/cidade
          }</Departamentos>
```

9) Retorna o código do departamento juntamente com seu endereço (rua e cidade) e nome dos seus empregados.

```
for $d in /Departamentos
return <Departamentos>{
                    $d/Departamento/id,
                      $d/Departamento/Endereco/rua,
                      $d/Departamento/Endereco/cidade,
                      <Empregados>{
                                $d/Departamento/Empregado/nome
                      }</Empregados>
          }</Departamentos>
```

10) Retorna o departamento juntamente com seu endereço (rua e cidade) e nomes dos seus empregados que moram na rua “José Lourenço”.

```
for $d in /Departamentos
where $d/Departamentos/Empregado/rua = "José Lourenço"
return <Departamentos>{
                    $d/Departamento/Endereco/rua,
```

```
                $d/Departamento/Endereco/cidade,
                <Empregados>{
                        $d/Departamento/Empregado/nome
                }</Empregados>
        }</Departamentos>
```

11) Retorna o código do departamento juntamente com seu endereço (rua e cidade) e nomes dos seus empregados que moram na rua “Abel Coelho”.

```
for $d in /Departamentos
where $d/Departamento/Empregado/rua = "Abel Coelho"
return <Departamentos>{
                $d/Departamento/id,
                  $d/Departamento/Endereco,
                  $d/Departamento/Empregado/nome
        }</Departamentos>
```

12) Retorna o nome dos empregados do departamento da cidade de “Fortaleza”.

```
for $d in /Departamentos
where $d/Departamento/Endereco/cidade = "Fortaleza"
return <Departamentos>{
                $d/Departamento/Empregado/nome
        }</Departamentos>
```

13) Retorna os empregados do departamento de “Automação”.

```
for $d in /Departamentos
where $d/Departamento/nome = "Automação"
return <Empregados>{
                $d/Departamento/Empregado
        }</Empregados>
```

14) Retorna o endereço e empregados de todos os departamentos.

```
for $d in /Departamentos
return <Departamentos>{
                $d/ Departamento/Endereco,
                $d/Departamento/Empregado
        }</Departamentos>
```

15) Retorna o código de todos os departamentos e para cada departamento lista o nome, rua de todos seus empregados.

```
for $d in /Departamentos
return <Departamentos>{
        $d/Departamento/id,
          for $e in $d/Departamento/Empregado
          return <Empregados>{
                $e/nome,
                $e/rua
                }</Empregados>
        }</Departamentos>
```

16) Retorna os empregados (rua, cidade e estado) dos departamentos que tem empregados chamado “Pedro”.

```
for $d in /Departamentos
where $d/Departamento/Empregado/nome = "Pedro"
return <Departamentos>{
        for $e in $d/Departamento/Empregado
        return <Empregados>{
                $e/rua,
                $e/cidade,
                $e/estado
                }</Empregados>
        }</Departamentos>
```

17) Retorna o endereço (rua e cidade) juntamente com o nome e estado de todos os empregados dos departamentos com código maior que 10500.

```
for $d in /Departamentos
where $d/Departamento/id > 10500
return <Departamentos>{
        <Endereco>{
                $d/Departamento/Endereco/rua,
                $d/Departamento/Endereco/cidade
        }</Endereco>,
        for $e in $d/Departamento/Empregado
        return <Empregados>{
                $e/nome,
                $e/estado
                }</Empregados>
        }</Departamentos>
```

18) Retorna o endereço (rua e cidade) juntamente com o nome de todos os empregados dos departamentos com código maior que 10500 ou com data de fundação igual a "10/05/2009".

```
for $d in /Departamentos
where $d/Departamento/id > 10500 or $d/Departamento/data_fundacao = xs.date("10/05/2009")
return <Departamentos>{
        <Endereco>{
                $d/Departamento/Endereco/rua,
                $d/Departamento/Endereco/cidade
        }</Endereco>,
        for $e in $d/Departamento/Empregado
        return <Empregados>{
                $e/nome
                }</Empregados>
        }</Departamentos>
```

19) Retorna a data de fundação e endereço do departamento "Financeiro" e para cada departamento retorne o nome e cidade dos empregados com códigos maiores que 200.

```
for $d in /Departamentos
where $d/Departamento/nome = "Financeiro"
return <Departamentos>{
        $d/Departamento/data_fundacao,
        $d/Departamento/Endereco,
        <Empregados>{
                for $e in $d/Departamento/Empregado
                where $e/empno >= 200
                return <Empregado>{
```

```
                    $e/nome,
                    $e/cidade
                    }</Empregado>
        }</Empregados>
        }</Departamentos>
```

20) Retorna o código do departamento e para cada departamento retorna o nome, rua, cidade e estado de todos seus empregados.

```
for $d in /Departamentos
return <Departamentos>{
        $d/Departamento/id,
         for $e in $d/Departamento/Empregado
         return $e/nome,
                $e/rua,
                $e/cidade,
                $e/estado
         }</Departamentos>
```

Anexo C

Consultas WFS executadas no *RelP* e no Deegree WFS

1) Retorna o código, nome e geometria de todos os postos.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>codigo</PropertyName>
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
	</Query>
</GetFeature>
```

2) Retorna o nome, geometria, rua e cidade do posto de código igual a "1250".

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
		<PropertyName>endereco/rua</PropertyName>
		<PropertyName>endereco/cidade</PropertyName>
		<Filter>
			<PropertyIsEqualTo>
				<PropertyName>codigo</PropertyName>
				<Literal>1250</Literal>
			</PropertyIsEqualTo>
		</Filter>
	</Query>
</GetFeature>
```

3) Retorna o código e geometria de todos os postos.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>codigo</PropertyName>
		<PropertyName>pluviometria</PropertyName>
	</Query>
</GetFeature>
```

4) Retorna o nome de todos os postos juntamente com suas pluviometrias (mês, valor).

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>pluviometria/mes</PropertyName>
		<PropertyName>pluviometria/valor</PropertyName>
	</Query>
</GetFeature>
```

5) Retorna o nome do posto com área maior que "5900", juntamente com suas pluviometrias (mês, valor).

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>pluviometria/mes</PropertyName>
                <PropertyName>pluviometria/valor</PropertyName>
                <Filter>
                        <PropertyIsGreaterThan>
                        <PropertyName>endereco/cidade/area</PropertyName>
                                <Literal>5900</Literal>
                        </PropertyIsGreaterThan>
                </Filter>
        </Query>
</GetFeature>
```

6) Retorna o nome e geometria do posto que apresente pluviometria menor que o mês "10".

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <PropertyIsLessThan>
                                <PropertyName>pluviometria/mes</PropertyName>
                                <Literal>10</Literal>
                        </PropertyIsLessThan>
                </Filter>
        </Query>
</GetFeature>
```

7) Retorna o nome e geometria do posto "São José" da "FUCEME".

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <And>
                        <PropertyIsEqualTo>
                                <PropertyName>nome</PropertyName>
                                <Literal>São José</Literal>
                        </PropertyIsEqualTo>
                        <PropertyIsEqualTo>
                                <PropertyName>orgao</PropertyName>
                                <Literal>FUCEME</Literal>
                        </PropertyIsEqualTo>
                        </And>
                </Filter>
        </Query>
</GetFeature>
```

8) Retorna o código e nome do posto do CEP igual a "59612300" ou do telefone "34910733".

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>codigo</PropertyName>
                <PropertyName>nome</PropertyName>
```

```
                <Filter>
                        <Or>
                        <PropertyIsEqualTo>
                                <PropertyName>endereco/cep</PropertyName>
                                <Literal>59612300</Literal>
                        </PropertyIsEqualTo>
                        <PropertyIsEqualTo>
                                <PropertyName>endereco/fone</PropertyName>
                                <Literal>34910733</Literal>
                        </PropertyIsEqualTo>
                        </Or>
                </Filter>
        </Query>
</GetFeature>
```

9) Retorna o código e nome dos postos entre os códigos “100” e “200”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>codigo</PropertyName>
                <PropertyName>nome</PropertyName>
                <Filter>
                        <PropertyIsBetween>
                        <PropertyName>codigo</PropertyName>
                            <LowerBoundary><Literal>100</Literal></LowerBoundary>
                            <UpperBoundary><Literal>200</Literal></UpperBoundary>
                         </PropertyIsBetween>
                </Filter>
        </Query>
</GetFeature>
```

10) Retorna o nome, endereço da cidade, pluviometrias e a geometria dos postos das cidades com área menor que ‘‘7000’’.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>endereco/cidade</PropertyName>
                <PropertyName>pluviometria</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <PropertyIsLessThan>
                                <PropertyName> endereco/cidade/area </PropertyName>
                                <Literal>7000</Literal>
                        </PropertyIsLessThan>
                </Filter>
        </Query>
</GetFeature>
```

11) Retorna o nome e geometria dos postos em que sua geometria esteja dentro da marcação retangular (*Bounding Box - BBOX*) especificada.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                <BBOX>
```

```
                    <PropertyName>geometria</PropertyName>
                    <gml:Box>
                        <gml:coordinates>-4.00,-40.00 -3.50,-39.50</gml:coordinates>
                    </gml:Box>
                </BBOX>
                </Filter>
        </Query>
</GetFeature>
```

12) Retorna o nome e geometria dos postos em que sua geometria intercepta espacialmente na geometria especificada.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                    <Intersects>
                    <PropertyName>geometria</PropertyName>
                    <gml:Point>
                        <coordinates>-3.916667,-39.966667</coordinates>
                    </gml:Point>
                    </Intersects>
                </Filter>
        </Query>
</GetFeature>
```

13) Retorna o nome dos postos em que sua geometria apresenta o mesmo tipo geométrico da geometria especificada.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <Filter>
                <Equals>
                    <PropertyName>geometria</PropertyName>
                    <Point><coordinates>0,0</coordinates></Point>
                </Equals>
                </Filter>
        </Query>
</GetFeature>
```

14) Retorna o nome e geometria dos postos em que sua geometria não seja disjunta com a marcação retangular especificada – (Ver requisição 12).

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                    <NOT>
                    <Disjoint>
                        <PropertyName>geometria</PropertyName>
                        <Box>
                            <coordinates>-4.00,-40.00 -3.50,-39.50</coordinates>
                        </Box>
                    </Disjoint>
                    </NOT>
```

```
                </Filter>
        </Query>
</GetFeature>
```

15) Retorna o nome e geometria dos postos em que sua geometria toca espacialmente o a geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <Touches>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                                <outerBoundaryIs>
                                <LinearRing>
                                <coordinates>55482.23359037499,932685.310856
                                        51792.79359037499,932912.810856
                                        51674.35359037499,932515.310856
                                        51544.143590375024,932078.310856
                                        49599.28359037498,925512.060856
                                        47775.54359037499,919372.560856
                                        47449.623590375006,918279.440856
                                        45808.45359037502,912738.8708560001
                                        45140.77359037497,910497.6208560001
                                        45004.94359037501,910041.6208560001
                                        45744.48359037499,909868.310856
                                        48668.97359037498,909195.190856
                                        50438.623590375006,917473.1208560001
                                        50803.29359037499,917410.810856
                                        50996.35359037499,918202.000856
                                        51108.95359037502,918699.6208560001
                                        55684.83359037497,927925.440856
                                        54440.553590375,928210.250856
                                        55318.413590374985,931999.1208560001
                                       55482.23359037499,932685.310856
                                </coordinates>
                                </LinearRing>
                               </outerBoundaryIs>
                        </Polygon>
                        </Touches>
                </Filter>
        </Query>
</GetFeature>
```

16) Retorna o nome e geometria dos postos em que sua geometria está contida espacialmente na geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <Within>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                        <outerBoundaryIs>
                                <LinearRing>
```

```
                                    <coordinates>55482.23359037,932685.310856
                                    51792.79359037,932912.810856
                                    51674.35359037,932515.310856 51544.14359038,932078.310856
                                    49599.28359037,925512.060856 47775.54359037,919372.560856
                                    47449.62359038,918279.440856 45808.45359038,912738.870856
                                    45140.77359037,910497.620856 45004.94359038,910041.620856
                                    45744.48359037,909868.310856 48668.97359037,909195.190856
                                    50438.62359038,917473.120856 50803.29359037,917410.810856
                                    50996.35359037,918202.000856 51108.95359038,918699.620856
                                    55684.83359037,927925.440856 54440.55359038,928210.250856
                                    55318.41359037,931999.120856
                                    55482.23359037,932685.310856</coordinates>
                                    </LinearRing>
                            </outerBoundaryIs>
                            </Polygon>
                            </Within>
                    </Filter>
            </Query>
</GetFeature>
```

17) Retorna o nome dos postos em que a geometria sobrepõe espacialmente a geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0"  outputFormat="GML2">
            <Query  typeName="POSTO">
                    <PropertyName>nome</PropertyName>
                    <Filter>
                            <Overlaps>
                            <PropertyName>geometria</PropertyName>
                            <Polygon>
                            <outerBoundaryIs>
                            <LinearRing>
                            <coordinates>55482.23359037,932685.310856  51792.79359037,932912.810856
                             51674.35359037,932515.310856       51544.14359038,932078.310856
                             49599.28359037,925512.060856       47775.54359037,919372.560856
                             47449.62359038,918279.440856       45808.45359038,912738.870856
                             45140.77359037,910497.620856       45004.94359038,910041.620856
                             45744.48359037,909868.310856       48668.97359037,909195.190856
                             50438.62359038,917473.120856       50803.29359037,917410.810856
                             50996.35359037,918202.000856       51108.95359038,918699.620856
                             55684.83359037,927925.440856       54440.55359038,928210.250856
                             55318.41359037,931999.120856       55482.23359037,932685.310856</coordinates>
                            </LinearRing>
                            </outerBoundaryIs>
                            </Polygon>
                            </Overlaps>
                    </Filter>
            </Query>
</GetFeature>
```

18) Retorna o nome dos postos em que sua geometria cruza espacialmente a geometria LineString especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
            <Query typeName="POSTO">
                    <PropertyName>nome</PropertyName>
                    <Filter>
                            <Crosses>
                            <PropertyName>geometria</PropertyName>
```

```
                        <LineString>
                                <coordinates>236331,901825 237853,909407</coordinates>
                        </LineString>
                        </Crosses>
                </Filter>
        </Query>
</GetFeature>
```

19) Retorna o nome dos postos em que sua geometria contém espacialmente a geometria LineRing especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <Filter>
                <Contains>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                        <outerBoundaryIs>
                        <LinearRing>
                        <coordinates>51108.95359037,918699.62085599
                         50996.35359037,918202.00085599   50803.29359037,917410.81085599
                         50438.62359038,917473.12085598   48668.97359037,909195.19085599
                         52231.52359038,908370.81085599   55984.13359037,907525.25085599
                         58244.77359038,906980.81085599   59527.73359037,906691.00085599
                         59931.81359037,908420.94085599   61718.14359037,916359.94085599
                         56192.09359037,917579.81085599   53636.07359037,918141.94085598
                         51108.95359037,918699.62085599</coordinates>
                        </LinearRing>
                        </outerBoundaryIs>
                         </Polygon>
                </Contains>
                </Filter>
        </Query>
</GetFeature>
```

20) Retorna os postos em que a geometria está mais além do raio de 100 metros partindo da geometria Point especificada.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <Filter>
                <Beyond>
                        <PropertyName>geometria</PropertyName>
                                <Point><coordinates>236331,901825</coordinates></Point>
                        <Distance>100</Distance>
                </Beyond>
                </Filter>
        </Query>
</GetFeature>
```

Anexo D

Consultas WFS executadas no *RelP* e no GeoServer

1) Retorna o código, nome e geometria de todos os postos.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>codigo</PropertyName>
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
	</Query>
</GetFeature>
```

2) Retorna o nome e geometria do posto de código igual a “1250”.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
		<Filter>
			<PropertyIsEqualTo>
				<PropertyName>codigo</PropertyName>
				<Literal>1250</Literal>
			</PropertyIsEqualTo>
		</Filter>
	</Query>
</GetFeature>
```

3) Retorna o código e nome dos postos entre os códigos “100” e “200”.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>codigo</PropertyName>
		<PropertyName>nome</PropertyName>
		<Filter>
			<PropertyIsBetween>
			<PropertyName>codigo</PropertyName>
			<LowerBoundary><Literal>100</Literal></LowerBoundary>
			<UpperBoundary><Literal>200</Literal></UpperBoundary>
			</PropertyIsBetween>
		</Filter>
	</Query>
</GetFeature>
```

4) Retorna o nome e rua dos postos que tem cep igual a “59612300”.

```
<GetFeature outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>rua</PropertyName>
		<Filter>
			<PropertyIsEqualTo>
			<PropertyName>cep</PropertyName>
```

```
                    <Literal>59612300</Literal>
                    </PropertyIsEqualTo>
                </Filter>
        </Query>
</GetFeature>
```

5) Retorna o nome, geometria, rua, cep e telefone de todos os postos com código maior que “2900”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <PropertyName>rua</PropertyName>
                <PropertyName>cep</PropertyName>
                <PropertyName>fone</PropertyName>
                <Filter>
                        <PropertyIsGreaterThan>
                                <PropertyName>codigo</PropertyName>
                                <Literal>2900</Literal>
                        </PropertyIsGreaterThan>
                </Filter>
        </Query>
</GetFeature>
```

6) Retorna o nome e geometria dos postos com código menor que “1300”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <PropertyIsLessThan>
                                <PropertyName>codigo</PropertyName>
                                <Literal>1300</Literal>
                        </PropertyIsLessThan>
                </Filter>
        </Query>
</GetFeature>
```

7) Retorna o nome e geometria do posto “São José” da rua “Domingos Olimpio”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <And>
                        <PropertyIsEqualTo>
                                <PropertyName>nome</PropertyName>
                                <Literal>São José</Literal>
                        </PropertyIsEqualTo>
                        <PropertyIsEqualTo>
                                <PropertyName>rua</PropertyName>
                                <Literal>Domingos Olimpio</Literal>
                        </PropertyIsEqualTo>
                        </And>
                </Filter>
```

```
        </Query>
</GetFeature>
```

8) Retorna o código e nome do posto que tenha telefone ou fax igual a “34910733”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>codigo</PropertyName>
                <PropertyName>nome</PropertyName>
                <Filter>
                        <Or>
                        <PropertyIsEqualTo>
                                <PropertyName>fone</PropertyName>
                                <Literal>34910733</Literal>
                        </PropertyIsEqualTo>
                        <PropertyIsEqualTo>
                                <PropertyName>fax</PropertyName>
                                <Literal>34910733</Literal>
                        </PropertyIsEqualTo>
                        </Or>
                </Filter>
        </Query>
</GetFeature>
```

9) Retorna todos os postos de código igual a “3”.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
        <Filter>
                <FeatureId fid="3"/>
        </Filter>
        </Query>
</GetFeature>
```

10) Retorna o nome e geometria dos postos em que sua geometria esteja dentro da marcação retangular (*Bounding Box* - *BBOX*) especificada.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                <BBOX>
                        <PropertyName>geometria</PropertyName>
                        <gml:Box>
                                <gml:coordinates>-4.00,-40.00 -3.50,-39.50</gml:coordinates>
                        </gml:Box>
                </BBOX>
                </Filter>
        </Query>
</GetFeature>
```

11) Retorna o nome e geometria dos postos em que sua geometria intercepta espacialmente na geometria especificada.

```
<GetFeature outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
```

```
		<PropertyName>geometria</PropertyName>
		<Filter>
			<Intersects>
			<PropertyName>geometria</PropertyName>
			<gml:Point>
				<coordinates>-3.916667,-39.966667</coordinates>
			</gml:Point>
			</Intersects>
		</Filter>
	</Query>
</GetFeature>
```

12) Retorna o nome dos postos em que sua geometria apresenta o mesmo tipo geométrico da geometria especificada.

```
<GetFeature version="1.0.0" outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<Filter>
		<Equals>
			<PropertyName>geometria</PropertyName>
			<Point><coordinates>0,0</coordinates></Point>
		</Equals>
		</Filter>
	</Query>
</GetFeature>
```

13) Retorna o nome e geometria dos postos em que sua geometria não seja disjunta com a marcação retangular especificada – (Ver requisição 12).

```
<GetFeature version="1.0.0" outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
		<Filter>
			<NOT>
			<Disjoint>
				<PropertyName>geometria</PropertyName>
				<Box>
					<coordinates>-4.00,-40.00 -3.50,-39.50</coordinates>
				</Box>
			</Disjoint>
			</NOT>
		</Filter>
	</Query>
</GetFeature>
```

14) Retorna o nome e geometria dos postos em que sua geometria toca espacialmente o a geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
	<Query typeName="POSTO">
		<PropertyName>nome</PropertyName>
		<PropertyName>geometria</PropertyName>
		<Filter>
			<Touches>
			<PropertyName>geometria</PropertyName>
			<Polygon>
```

```
                                <outerBoundaryIs>
                                <LinearRing>
                                <coordinates>55482.23359037499,932685.310856
                                        51792.79359037499,932912.810856
                                        51674.35359037499,932515.310856
                                        51544.143590375024,932078.310856
                                        49599.28359037498,925512.060856
                                        47775.54359037499,919372.560856
                                        47449.623590375006,918279.440856
                                        45808.45359037502,912738.8708560001
                                        45140.77359037497,910497.6208560001
                                        45004.94359037501,910041.6208560001
                                        45744.48359037499,909868.310856
                                        48668.97359037498,909195.190856
                                        50438.623590375006,917473.1208560001
                                        50803.29359037499,917410.810856
                                        50996.35359037499,918202.000856
                                        51108.95359037502,918699.6208560001
                                        55684.83359037497,927925.440856
                                        54440.553590375,928210.250856
                                        55318.413590374985,931999.1208560001
                                       55482.23359037499,932685.310856
                                </coordinates>
                                </LinearRing>
                                </outerBoundaryIs>
                        </Polygon>
                        </Touches>
                </Filter>
        </Query>
</GetFeature>
```

15) Retorna o nome e geometria dos postos em que sua geometria está contida espacialmente na geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <PropertyName>geometria</PropertyName>
                <Filter>
                        <Within>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                        <outerBoundaryIs>
                                <LinearRing>
                                <coordinates>55482.23359037,932685.310856
                                51792.79359037,932912.810856
                                51674.35359037,932515.310856 51544.14359038,932078.310856
                                49599.28359037,925512.060856 47775.54359037,919372.560856
                                47449.62359038,918279.440856 45808.45359038,912738.870856
                                45140.77359037,910497.620856 45004.94359038,910041.620856
                                45744.48359037,909868.310856 48668.97359037,909195.190856
                                50438.62359038,917473.120856 50803.29359037,917410.810856
                                50996.35359037,918202.000856 51108.95359038,918699.620856
                                55684.83359037,927925.440856 54440.55359038,928210.250856
                                55318.41359037,931999.120856
                                55482.23359037,932685.310856</coordinates>
                                </LinearRing>
                        </outerBoundaryIs>
                        </Polygon>
                        </Within>
```

```
                </Filter>
        </Query>
</GetFeature>
```

16) Retorna o nome dos postos em que a geometria sobrepõe espacialmente a geometria Polygon especificada pelas coordenadas.

```
<GetFeature version="1.0.0"  outputFormat="GML2">
        <Query  typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <Filter>
                        <Overlaps>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                        <outerBoundaryIs>
                        <LinearRing>
                        <coordinates>55482.23359037,932685.310856  51792.79359037,932912.810856
                         51674.35359037,932515.310856       51544.14359038,932078.310856
                         49599.28359037,925512.060856       47775.54359037,919372.560856
                         47449.62359038,918279.440856       45808.45359038,912738.870856
                         45140.77359037,910497.620856       45004.94359038,910041.620856
                         45744.48359037,909868.310856       48668.97359037,909195.190856
                         50438.62359038,917473.120856       50803.29359037,917410.810856
                         50996.35359037,918202.000856       51108.95359038,918699.620856
                         55684.83359037,927925.440856       54440.55359038,928210.250856
                         55318.41359037,931999.120856       55482.23359037,932685.310856</coordinates>
                        </LinearRing>
                        </outerBoundaryIs>
                        </Polygon>
                        </Overlaps>
                </Filter>
        </Query>
</GetFeature>
```

17) Retorna o nome dos postos em que sua geometria cruza espacialmente a geometria LineString especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <Filter>
                        <Crosses>
                        <PropertyName>geometria</PropertyName>
                        <LineString>
                                <coordinates>236331,901825 237853,909407</coordinates>
                        </LineString>
                        </Crosses>
                </Filter>
        </Query>
</GetFeature>
```

18) Retorna o nome dos postos em que sua geometria contém espacialmente a geometria LineRing especificada pelas coordenadas.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <PropertyName>nome</PropertyName>
                <Filter>
```

```
                <Contains>
                        <PropertyName>geometria</PropertyName>
                        <Polygon>
                        <outerBoundaryIs>
                        <LinearRing>
                        <coordinates>51108.95359037,918699.62085599
                         50996.35359037,918202.00085599    50803.29359037,917410.81085599
                         50438.62359038,917473.12085598    48668.97359037,909195.19085599
                         52231.52359038,908370.81085599    55984.13359037,907525.25085599
                         58244.77359038,906980.81085599    59527.73359037,906691.00085599
                         59931.81359037,908420.94085599    61718.14359037,916359.94085599
                         56192.09359037,917579.81085599    53636.07359037,918141.94085598
                         51108.95359037,918699.62085599</coordinates>
                        </LinearRing>
                        </outerBoundaryIs>
                         </Polygon>
                </Contains>
                </Filter>
        </Query>
</GetFeature>
```

19) Retorna os postos em que sua geometria está dentro do raio de 10 metros partindo da geometria Point especificada.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <Filter>
                        <DWithin>
                                <PropertyName>geometria</PropertyName>
                                <Point><coordinates>236331,901825</coordinates></Point>
                        <Distance units="meter">10</Distance>
                        </DWithin>
                </Filter>
        </Query>
</GetFeature>
```

20) Retorna os postos em que a geometria está mais além do raio de 100 metros partindo da geometria Point especificada.

```
<GetFeature version="1.0.0" outputFormat="GML2">
        <Query typeName="POSTO">
                <Filter>
                <Beyond>
                        <PropertyName>geometria</PropertyName>
                                <Point><coordinates>236331,901825</coordinates></Point>
                        <Distance>100</Distance>
                </Beyond>
                </Filter>
        </Query>
</GetFeature>
```